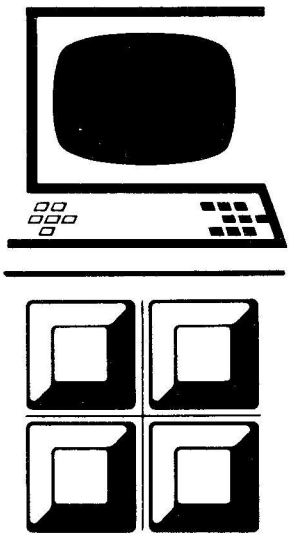
System 6300 Administrator's Guide

SOFTWARE RELEASE FE02



87601213A

MOTOROLA
Information Systems

Document Number: S6000-50-1A
Stock Number: 87601213A

First Edition: January 1984
B-09-00410-01
Issue A: 15 March 1984
Change 1: 15 June 1984

Specifications subject to change.
Copyright 1984, 1983 by Convergent Technologies, Inc.

Selected Series 6000 Publications

- Series 6000 Operating System Reference Manual (Volumes 1 and 2, S6000-50-6 and S6000-50-7)--Explains how to use the UNIX-derived system commands, library routines, and I/O functions.

- System 6300 Software Installation Guide (S6000-40-1)--Describes how to prepare the System 6300 software for first time use.

- System 6300 Hardware Installation and User's Guide (S6000-22-1)--Describes how to install the System 6300 hardware.

- System 6300 Administrator's Guide (S6000-50-1)--Provides information on how to start and stop the System 6300, system management, and system maintenance.

- Series 6000 Operating System Programmer's Guide (S6000-50-5)--Explains how to use the UNIX-derived command language, the C programming language, and the utility features associated with the UNIX-derived operating system.

- Series 6000 UNIVIEW General Functions (S6000-50-2)--Describes how to access Series 6000 application software using UNIVIEW menus and commands.

- Series 6000 UNIVIEW Supervisory Functions (S6000-50-3)--Details how to implement UNIVIEW supervisory functions such as altering user profiles and passwords, and configuring the system.

- Introduction to UNIVIEW (S6000-50-8)--Provides a survey of UNIVIEW's different functions.

- Series 6000 UNIVIEW Programmer's Guide (S6000-50-4)--Provides information on how to program UNIX-derived applications programs in the UNIVIEW environment.

- Series 6000 COBOL Programmer's Guide (S6000-45-1)--Describes how to use the COBOL language with the Series 6000 computer system.

- Series 6000 COBOL/SIBOL Runtime Manual (S6000-45-2)--Explains how to execute a COBOL or SIBOL computer program on the Series 6000 computer system.

- Pascal Programmer's Manual (S6000-45-6)--Describes the Pascal programming language.

- Series 6000 Pascal Extensions (S6000-45-4)--Explains the differences between standard Pascal and the Series 6000 Pascal.

PREFACE

This manual describes the responsibilities of the administrator of a System 6300 Computer System. It gives procedures to be followed and describes programs and files to be used.

Readers should be familiar with basic operating system interaction as described in the first part of the Series 6000 Operating System Programmer's Guide.

This manual describes programs and files in simple terms, sufficient for the administrator's purposes. For detailed descriptions, refer to the Series 6000 Operating System Reference Manual.

This manual contains the following sections:

- Section 1, Introduction, explains what an administrator is and does.
- Section 2, Administrative Interaction, describes the special operating modes that give the administrator complete access to the system.
- Section 3, Adding New Peripheral Devices, tells how to make support terminals and printers.
- Section 4, Using Disks, tells how disks are organized, how to initialize a disk and divide it into operating system slices (partitions), and how to create and maintain a file system.
- Section 5, User Support, tells how allocate and deny users access to the system and its resources.
- Section 6, Backups and Restores, tells how to efficiently preserve offline copies of files against accidental loss.
- Appendix A, File System Concepts, explains the data structures and concepts behind a file system.
- Appendix B, Init and Getty, is a programmer's view of the two programs that activate a terminal.

- Appendix C, System Accounting, tells how to use user accounting programs.

- Appendix D, Lp Spooling System, tells how to configure and maintain **lp** and its related programs. **Lp** is a powerful and flexible alternative to the standard **lpr** program.

Portions of this manual are excerpts of AT&T documents that describe the UNIX-derived operating system.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in paragraph (b) (3) (B) of the Rights in Technical Data and Computer Software clause in DAR 7-104.9 (a).

Motorola, Inc.
10700 North De Anza Boulevard
Cupertino, California 95014

Contents

SECTION 1: INTRODUCTION

The System 6300 administrator configures and allocates System 6300 operating system resources. The administrator accesses the system in ways other users can't and controls the way other users use the system.

The administrator has the following specific responsibilities:

- Giving and denying other users access to the system.
- Specifying what disks and disk files users can access.
- Preparing new disks for use by the system
- Telling the system how to use new terminals and printers.
- Performing routine backup of disk files to prevent accidental loss of data.
- Starting the system running and turning the system off.

SECTION 2: ADMINISTRATIVE INTERACTION

The System 6300 administrator uses the operating system in ways that ordinary users can't. This is necessary so that the administrator can manage the system and its resources.

The administrator has three special ways to interact with the operating system. Note that the first is not exclusive of the other two.

- Superuser status. When using the operating system as superuser, you can ignore restrictions on file access and allowable commands.
- Single-user mode. When the operating system is in this mode, only one terminal is usable. Single-user mode is used for procedures that require an absence of normal disk activity. The single user in single-user mode has superuser status.
- Standalone shell. This is a program that provides some administrative commands when the operating system is not running. The single user running the standalone shell has superuser status.

WARNING

Do not halt, reset, or turn off a System 6300 Computer System unless the operating system is not running or is running in single-user mode.

SUPERUSER STATUS

Superuser status removes important operating system restrictions. The administrative commands in this manual require superuser status. The operating system gives the superuser three exemptions from normal restrictions:

- File read and write permissions do not apply to the superuser. The superuser can write to or read from any ordinary or special file. The superuser can create a file in or delete a file from any directory.

- Certain commands are executable only by the superuser.
- Some commands have built in safeguards or restrictions on the way they are used. Some safeguards and restrictions do not apply to the superuser.

When the operating system is running normally, there are two ways to obtain superuser status.

- Log in as user "root."
- Use the superuser program, **su**.

Both accesses to superuser status require knowledge of root's user password. Consider root's password sensitive information. Change it regularly.

When the operating system is not running normally (single-user mode or the standalone shell), the sole user normally has superuser status.

The shell changes its prompt to remind you that you are superuser. Normally the default prompt is a dollar sign ($). When the superuser runs the shell, the default prompt is a pound sign (#).

The Root User

In the password file, **/etc/passwd**, the user called root has numeric user ID 0; this identifies root as the superuser. Under no circumstances change the name, numeric user ID, or numeric group ID of this user. Root should be the first user in the file.

Root's password is a sensitive piece of information: anyone who knows it can become superuser. When your system is first booted, root has no password and anyone can become superuser. To provide an initial or changed password, run **passwd**:

```
passwd
```

Passwd prompts for the old password once (if there is one) and the new password twice.

Root's home directory is /, but this directory should not have any more files in it than necessary.

Example -- Changing a Password

A user has changed his own password, then forgotten the new password. There is no way to reverse password encryption, so the valid password is lost forever. The only solution is for the user to get a new password, but only the user himself can change his own password and even then only if he knows his existing password. Fortunately, these restrictions don't apply to the superuser.

The administrator's input is in bold, the computer's responses in normal type, ### indicates unechoed input, and f indicates Control-D.

```
$ passwd walter
permission denied
$ login walter
password: ###
login incorrect
login: root
password: ###
# passwd walter
Changing password for walter
New password: ###
Retype new password: ###
#
```

The Su Program

To become superuser while logged in as an ordinary user, use the superuser program:

```
su
```

Su will prompt for a password; enter root's password. If the password is verified, **su** runs the shell with its numeric user ID set to 0, giving the shell the same status as a shell run by root.

To return to normal user status, terminate the **su** shell with Control-D . You can also return to normal user status by using **su** with your own (or any other) user name, but this doesn't terminate execution of the superuser shell.

Example -- Using the Su Command

A system administrator changes root's password while logged in as an ordinary user. The administrator's input is in bold, the computer's responses in normal type, ### indicates unechoed input, and f indicates Control-D.

```
$ su
Password: ###
# passwd
Changing password for root
Old password: ###
New Password: ###
Retype new password: ###
# f
```

SINGLE-USER MODE

Single-user mode prevents ordinary users from communicating with the system. This prevents normal activity that might interfere with disk backup and maintenance.

There are two ways the operating system can go to its single-user mode.

- By commands from the system administrator.
- Automatically on start up if the operating system decides it is not safe to go to multiuser mode.

Both methods indirectly use the **telinit** command, a command that sends signals to the process initialization process, **init**. Do not change to single-user mode by using **telinit** directly: **telinit** does not give user programs a chance to terminate cleanly.

When the operating system is in single-user mode, only one terminal is usable: the terminal that was used to take the system to single-user mode. The user using this terminal has superuser status.

Taking the Operating System to Single-User Mode

To take the operating system to its single-user mode:

1. Make / your working directory.
2. Run **shutdown**:

   /etc/shutdown grace

   where grace is the number of seconds the users get to log out by themselves; if grace is omitted, the users get 60 seconds. **Shutdown** runs **wall** to warn the users, **killall** to terminate the users, and **init** to change the system mode; this process takes about a minute. See the Introduction to System 6300 Operating System.

Here's an example of going to single-user mode. A system administrator logged in as an ordinary user takes the system to single-user mode, giving the users two minutes to log out. The administrator's input is in bold, the computer's responses in normal type, ### indicates unechoed input, and f indicates Control-D.

```
$ su
Password:###
# cd /
# /etc/shutdown 120

SHUTDOWN PROGRAM

Aug 20 09:02 1983

Do you want to send your own message? (y or n):  y
Type your message followed by ctrl d....

Taking system down for weekly backups.  You have 2 minutes
to log out.
f
Broadcast message from adm....

Taking system down for weekly backups.  You have 2 minutes
to log out.
Broadcast message from adm....

SYSTEM BEING BROUGHT DOWN NOW ! ! !
Busy out (push down) the appropriate
phone lines for this system.

Do you want to continue? (y or n):  y
Process accounting stopped
Error logging stopped

All currently running processes will now be killed.

unmounting /dev/fp003

Wait for `INIT: SINGLE-USER MODE' before halting.

INIT: SINGLE-USER MODE

#
```

There is a two-minute delay after the first broadcast message and a one-minute delay after the second.

Automatically Going to Single-User Mode

The operating system has a start up sequence that is executed whenever the system is turned on or reset. The start up sequence includes a check of the file systems. The file system check can have three outcomes.

- Nothing is wrong with any file system. The operating system goes to multiuser mode.
- One or more file systems is corrupt, but it is possible to fix them without destroying any data. The operating system fixes the corrupt file systems, then goes to multiuser mode. (However, if it had to fix the root file system, the operating system reboots itself, starting all over.)
- One or more file systems is so corrupt that no automatic fix is evident. The operating system goes to administrator mode.

When the operating system is in administrator mode, designated terminals prompt for the administrator to log on. If an ordinary user logs in, the system promptly logs him or her off. If root logs in, the system switches to single-user mode. The single working terminal in single-user mode is the terminal on which root logged in.

The system administrator specifies which terminals are to be active in administrator mode when he configures the terminals. See the section on adding new peripheral devices.

Taking the Opeating System to Multiuser Mode

To return to normal multiuser mode, terminate the shell. Press Control-D in response to a shell prompt. The system will prompt "Run level?" Enter **2** and press the RETURN key.

STANDALONE SHELL

The standalone shell provides a limited operating system environment when you cannot or should not boot the system. All commands are implemented on a diskette, so access to the fixed disk is avoidable.

There are three restrictions on standalone shell command:

- No files can be written to. Existing files can be read.

- No input/output redirection, is allowed. The |, <, and > constructs are illegal.

- Only the following commands are implemented:

cat	fsdb	iv
dd	ls	volcopy
fsck	mkfs	

To run the standalone shell:

1. Take the operating system to single-user mode, if it is running.

2. Insert the standalone diskette or cartridge in the drive.

3. Reset the system.

SECTION 3: ADDING NEW PERIPHERAL DEVICES

This section describes the operating system changes required by new peripheral devices. This consists of changing certain configuration files that the operating system uses and of making sure that the operating system responds to the change.

CONFIGURING A NEW TERMINAL

Each new terminal requires the following actions:

1. Determine the new terminal's number.
2. Create an entry in the configuration file for the **init** program.
3. Create an entry in the file that lists terminal types.
4. Make sure that **init** rereads its configuration file.

It is important to distinguish between RS-232 terminals and RS-422 terminals. Those terms actually describe the kind of communication link between the terminal and the System 6300 Computer System. Each RS-232 line supports a single terminal. The RS-422 line supports up to eight terminals. Most terminals can only be used on an RS-232 line. Motorola TM30 Workstations and graphics terminals can be used on either kind of line.

The Terminal Number and the Console

Each terminal has a three-digit decimal number. Terminal numbers start from **000**. Note that a terminal number is always expressed in three digits, even though the terminal numbers do not currently exceed 027. Numbers are expressed this way so that system programs can be the same on System 6600 and System 6300 systems.

Terminal numbers 000 through 010 designate RS-232 terminals. The terminal number indicates the line used.

Terminal numbers 020 through 027 designate RS-422 terminals. An RS-422 terminal does not automatically get a certain number: the number is assigned when you turn the terminal on. Turning the terminal on gives it the lowest RS-422 terminal number not already in use. Turning the terminal off frees its number; this number may then be appropriated by some other terminal. Thus the only way to make sure that an RS-422 terminal gets a specific number is to control the order in which the RS-422 terminals are turned on. Normally it is not necessary to make sure that a terminal has a specific terminal number. But note how many RS-422 terminals are in use so you will know what terminal numbers will be allocated.

Certain important system messages are sent to the system console, **/dev/console**. This is simply a link to terminal 000.

Configuring Getty

The text file **/etc/gettydefs** is used by **getty**, the operating system's terminal initializer. Each entry specifies a set of communication options and a log-in message. Each set of similar terminals connected to the system requires two entries: one for multiuser mode and one for administrator mode.

As distributed, **/etc/gettydefs** contains three entries:

- The **9600** entry, which defines communication options suitable for an RS-232 9600 baud terminal and a multiuser mode log-in message.
- The **C9600** entry, which defines communication options suitable for an RS-232 9600 baud terminal and an administrator mode log-in message.
- The **RS422** entry, which defines communication options suitable for an RS-422 terminal and a multiuser mode log-in message.

Each entry in **/etc/gettydefs** is a line of the form

label#ioptions#foptions#message#next

where

label — identifies the entry. The only strict rule is that label be unique in the file. A common convention labels a multiuser mode entry with its baud rate and an administrator mode entry with **C** followed by the baud rate; use this convention only if it's convenient.

ioptions is a list of communications options for **getty** to apply when it first opens the terminal. Specify options with the symbolic constants described under **termio**(7) in the Series 6000 Operating System Reference Manual. Symbolic constants are separated from each other by spaces or tabs.

foptions is a list of communications options for **getty** to apply before calling **login** (that is, after a user first enters a log-in name). Foptions contains the same kind of information as does ioptions.

message is text to print when the terminal is first opened. The text should end with "login: ".

next indicates another entry to use if **getty** receives a break while it's using this entry. If you don't know one or more of a terminal's communication options in advance (most often the speed), use the next fields to form a circular linked list of entries. A user can then select the right entry by pressing the BREAK key until a log-in message appears.

To include nongraphic characters in the entry, use one of the following sequences:

\n	newline
\t	tab
\v	vertical tab (Control-K)
\b	backspace
\r	carriage return
\f	form feed
****xxx	where xxx is a 1- to 3-digit octal number

A backslash (****) followed by any character not mentioned above just stands for the second character. Thus you enter **** to get a ****.

There should be only one difference between a multiuser mode entry and the corresponding administrator mode entry. The administrator mode entry should have a message that reminds the user that the system is in administrator mode.

Check the correctness of the new **/etc/gettydefs** after modifying it. The following command finds errors:

```
/etc/getty -c /etc/gettydefs
```

Configuring Init

The text file **/etc/inittab** is used by **init**, the master process spawner. Each new terminal requires that the init table be modified so that the operating system monitors the terminal for attempted log ins. Use a text editor to modify **/etc/initab.**

Each terminal requires a line in **/etc/inittab** of the form

ttt**:23:respawn:/etc/getty tty**ttt def

where

ttt is the terminal number.

def indicates a multiuser mode definition in **/etc/gettydefs.**

Each terminal that is to be active in administrator mode requires a line in **/etc/inittab** of the form

C**ttt:6:respawn:/etc/getty tty**ttt def

where

ttt is the terminal number.

def indicates an administrator mode definition in **/etc/gettydefs.**

Normally, only terminal 000 (the console) is active in administrator mode. If you need any RS-422 terminals to be active in administrator mode, make all RS-422 terminals active: you cannot know in advance which specific RS-422 terminal gets which specific terminal number.

Configuring Terminal Type

The terminal type file, **/etc/ttytype,** lists the kind of terminal represented by each terminal number. Use an editor to modify this file.

Each entry in **/etc/ttytype** is a line of the form

type **tty**nnn

where

type is a terminal type. A TM30 Workstation is **pt.** For other codes, search the file **/etc/termcap.** To add new terminals to this file see the discussion of

termcap(4) in the Series 6000 Operating System Reference Manual.

nnn is the three-digit terminal number.

Rereading Terminal Configuration Files

Modification of **/etc/gettydefs** or **/etc/inittab** does not require a reboot of the operating system. For **/etc/inittab,** it is only necessary to make **init** reread it. For **/etc/gettydefs,** it may be necessary to kill **getty** instances made obsolete by the change.

When the operating system is running normally, **init** rereads **/etc/inittab** every time a user logs off and every five minutes. If this is not soon enough and it is inconvenient to reboot, the following command tells **init** to reread **/etc/inittab.**

telinit q

WARNING

Use the **telinit** command carefully and precisely. The wrong parameter will stop the operating system suddenly and painfully.

Getty uses new entries in **/etc/gettydefs** without any prompting, but an instance of **getty** acting on an obsolete **/etc/gettydefs** entry can tie up a terminal. If a terminal remains unusable after a change to **/etc/gettydefs** try the following two steps.

1. Discover the process number of the **getty** monitoring the terminal:

   ps -txxx

   where xxx is the three-digit terminal number.

2. Terminate **getty:**

   kill n

   where n is the process number displayed by **ps.**

Example

A System 6300 Computer System currently has a single terminal: a TM30 Workstation connected to an RS-232 line. The system administrator adds four new terminals: a Datamedia 2500,

connected to an RS-232 line; and three TM30 Workstation terminals, connected to the RS-422 line.

In the following example, the system administrator has just connected the new terminals and has logged in as root. All new terminals are on the RS-422 line or run at 9600 baud, so there is no need to modify **/etc/gettydefs.** None of the new terminals are to be active in administrator mode. The administrator does not know the terminal type that corresponds to the new RS-232 terminal, so he checks **/etc/termcap.**

```
# fgrep datamedia /etc/termcap
D0|dm1520|dm1521|1521|1520|datamedia 1520:\
D1|dm1521|1521|datamedia 1521:\
D2|dm2500|datamedia2500|2500|datamedia 2500:\
D3|dm3025|datamedia 3025a:is=\EQ\EU\EV:\
D4|3045|dm3045|datamedia 3045a:is=\EU\EV:\
D5|dt80|dmdt80|dm80|datamedia dt80/1:\
D6|dt80132|dmdt80132|datamedia dt80/1 in 132 char mode:\
# ed /etc/inittab
32
1,$p
000:23:respawn:/etc/getty tty000 9600
C000:6:respawn:/etc/getty tty000 C9600
$a
001:23:respawn:/etc/getty tty001 9600
020:23:respawn:/etc/getty tty020 RS422
021:23:respawn:/etc/getty tty021 RS422
022:23:respawn:/etc/getty tty022 RS422
.
w
610
e /etc/ttytype
6
$p
pt 000
$a
dm2500 001
pt 020
pt 021
pt 022
.
w
34
q
# /etc/telinit q
#
```

Removing Terminals

Be absolutely sure that **/etc/inittab** does not refer to any terminal numbers not in use. **Init** may experience difficulties

bringing the system to multiuser state if it tries to open nonexistent terminals.

Note that removing an RS-422 terminal always abandons the highest terminal number. It does not matter which RS-422 terminal you remove.

Configuring Modems

Modems require special treatment because they allow two kinds of communication: dial in connections, and dial out connections. Dial in connections are established from outside the system. Dial out connections are established by the operating system.

To permit dial in connections, the line must be monitored by **getty**, just like the directly connected terminals. When the modem responds to another modem, **getty** wakes up and starts the log in process.

When there is a dial out connection, a user program such as **cu**, **ct**, or **uucp** opens the terminal line and uses the modem to establish a communication link. When the user program begins, **getty** must not be monitoring the line, or it will interfere with the user program.

To establish this kind of dual use, do not give the modem a normal multiuser mode entry in **/etc/inittab**. Instead, make the following entry in **/etc/inittab**.

ttt:3:respawn:/etc/getty ttyttt def

where

ttt is the terminal number of the modem's line.

def indicates a multiuser mode definition in **/etc/gettydefs**.

No reboot or message to **init** is necessary initially. Dial out connections are possible; dial in connections are not. To allow dial in connection, do

telinit 3

To banish dial in connections again, follow the following procedure.

1. Look at your list of users to make sure no one is logged in over the modem lines:

   who

2. Tell the system to banish **getty** from the modem lines:

telinit 2

WARNING

Use the **telinit** command carefully and precisely. The wrong parameter will stop the operating system suddenly and painfully.

Note that when the operating system is booted, dial in connection are impossible until you allow them.

If you lose track of which **telinit** you issued last, do

who -r

The number printed after "run level" is the number you passed to **telinit**.

PRINTERS

The operating system provides two incompatible print spooling programs: **lp** and **lpr**. **Lp** supports multiple printers and has advanced features. **Lpr** is simple but adequate for a system with a single printer. Appendix D tells how to configure **lp**; the remainder of this subsection tells how to configure **lpr**.

Lpr can use two kinds of printers:

- Centronics-compatible parallel printer. This is connected to the special parallel printer line.
- Serial printer. This is connected to any RS-232 terminal line.

The parallel printer line is associated with the special file **/dev/plp**.

An RS-232 line used by a receive-only printer must not be monitored for log ins. Examine **/etc/inittab** and verify the absence of any reference to the line you intend to use. Terminal line usage is discussed earlier in this section.

To make **lpr** use the correct line, associate the special file **/dev/lp** with the correct line. If the printer is connected to the parallel printer line, do

ln /dev/plp /dev/lp

If the printer is connected to an RS-232 terminal line, do

ln /dev/ttyttt **/dev/lp**

where

ttt is the three-digit terminal number.

As shipped, the System 6300 operating system has **/dev/lp** associated with the parallel printer line.

Use **ls** to verify that a **/dev/lp** is associated with the correct line. Two special files are associated with the same line if they have they same i-number. To see if **/dev/lp** has the same i-number as **/dev/plp**, do

ls -li /dev/plp /dev/lp

This produces a listing like this:

```
3438 crwxrwxrwx 2 root root      6, 0 /dev/plp
3438 crwxrwxrwx 2 root root      6, 0 /dev/lp
```

Only the i-number (first item on each line) and the special file name (last item on each line) are of interest. Verify that the two i-numbers are identical; if they aren't, **/dev/lp** is not associated with **/dev/plp.**

To see if **/dev/lp** has the same i-number as an RS-232 terminal line do

ls -li /dev/ttyttt **/dev/lp**

where

ttt is the three-digit terminal number.

This produces a listing much like that of the other version of the **ls** command. Again, compare i-numbers: they must be the same or **/dev/lp** is not associated with the indicated line.

If a serial printer is used and the printer's communication requirements do not match the operating system's defaults, you must arrange for the operating system to set and hold terminal options. To do that, add the following two lines at the end of **/etc/rc:**

sleep 2000000 > /dev/lp &
stty options **< /dev/lp**

where options is a list of valid **stty** options: see **stty**(1) in the Series 6000 Operating System Reference Manual.

CONFIGURING CALL-UP DEVICES

When your system calls a remote system, nuucp opens the terminal line and makes a communications link between the two systems. To enable nuucp to establish this link, you must first alter the **/etc/inittab** file for tty00l on each system. For the initiating system, create or alter the line for tty00l so that it looks like this:

```
00l:4:respawn:/etc/getty tty00l 9600
```

For the responding system, create or alter the line for tty00l so that it appears this way:

```
00l:2:respawn:/etc/getty -t 30 tty00l 9600
```

If you wish to attach a password to the nuucp utility type

```
passwd nuucp
```

and enter the password you want to use for this utility. When the password prompt appears the second time, reenter the password to make certain you spelled it correctly the first time.

For both the initiating and responding systems you must set the permission modes to read and write permission for owner, group, and others. For each system then, write

```
chmod 0666 /dev/ttyxxx
```

where **xxx** is the number of the terminal that you are using for communication.

SECTION 4: USING DISKS

This section describes use of the System 6300 disks. It describes how disks are organized and how the system administrator makes the disk's storage capacity available to users.

There are two basic kinds of disks: fixed, and removable (diskettes). Fixed and removable disks are organized in a very similar way; except where noted, the following information applies to both kinds.

The procedure that installs the operating system initializes the fixed disk and installs the reserved area (slice 0), the root file system (slice 1), and the swap slice (slice 2). Thus you do not initialize the fixed disk or create the root file system unless you accidentally undo some part of their configuration.

This section discusses use of the **iv** command. For a complete reference on this command, see the the Series 6000 Operating System Reference Manual.

DISK ORGANIZATION

This subsection explains how disks are organized. A few concepts are introduced here that express how the disk is divided into manageable units and what purposes the units serve.

A disk is divided into 1 to 16 slices. A slice is simply a section of the disk that is used as a unit. Slices are numbered 0 to 15. Slice 0, also called the Reserved Area, is preassigned to hold data required to manage the disk; there is normally at least one additional slice.

If a disk is used to boot the operating system (as is the fixed disk), slices 1 and 2 also have preassigned purposes. Slice 1 contains the root file system (see below). Slice 2 provides swap space; the size of this slice places a practical limit on the program size.

File Systems

An important use of a slice is to contain a file system. A file system consists of the files plus the data structures the operating system kernel requires to support file uses. Normally, all the slices on the fixed disk except for 0 (the reserved area) and 2 (the swap area) contain file systems.

To be used, a file system must be mounted to give it a place in the directory hierarchy. The root file system in slice 1 is permanently mounted; other file systems are mounted by the **mount** command or by an automatic procedure that is executed when the system is booted.

Users use the mounted file systems as a single hierarchy of directories and files. For the most part, the division of file hierarchy into slices is not apparent to users and disk organization is not an ordinary user's problem.

Be careful not to destroy the root file system. Doing so renders the system unusable.

Swap Slice

The size of the swap slice determines the total limit on program memory usage. Slice two on the disk from which the operating system was booted is always the swap slice; thus the swap slice on the fixed disk is **/dev/fp002.** A sign that the swap slice is too small is the failure of a large number of commands with messages like "not enough space." Experience will tell you how much swap space you need, but a good rule of thumb is that each terminal uses up 1 megabyte. In any case the swap slice should be a little larger than the system's random access memory (RAM) so that the RAM is fully utilized.

Do not write a file system or any other data into a swap slice.

Other Slices

A disk, especially a removable disk, can see uses other than holding file systems or the swap slice. Some data base systems manage slices directly, without using a file system. Removable disks are often used for routine file backups and for transfer of data between systems; such disks contain a single slice with no file system.

Unlike the swap slice or a file system, which are managed by the operating system kernel, a plain slice is managed by the programs that use it.

INITIALIZING AND CONFIGURING DISKS

This subsection explains procedures that initialize a disk and divide it into slices. Normally you would use these procedures on new removable disks.

If you use any of these procedures on a disk already in use (for example, to adjust the slice sizes on the fixed disk) assume that all data on the disk will be lost. In the case of the fixed disk, do a complete backup before adjusting slice sizes or otherwise changing the disk configuration; see the section on backups and restores.

Initializing and configuring a disk requires the following steps:

1. Determine the dimensions of the disk.
2. Plan the disk's division into slices and how the slices are to be used.
3. Create a disk description file.
4. Run the disk initialization utility, **iv**.

Use this procedure both to initialize a new disk and to specify new slice boundaries on an old disk.

Before applying this procedure to an old disk, copy or backup the disk! It is possible to rearrange slice boundaries without damaging all existing slices, but do not count on your ability to do this.

After initializing a removable disk, use one of the following commands just before removing the disk from the drive:

dismount -f

For instruction on using **dismount**, see "Using Removable Disks," below.

Determining Disk Dimensions

The prototype disk description files in **/usr/lib/iv** describe the various disks commonly used with the System 6300. Use **cat** or some other text file utility to examine the file for your disk.

cat *file*

where

file is the prototype description file name.

The prototype description file gives physical characteristics of the disk. Of interest are heads, cylinders, and sectors. If sectors is an odd number, the disk has a sector on each track reserved as a bad block alternate: subtract one from sectors before using it. Calculate the total number of 1024-byte logical blocks the disk can contain:

heads x cylinders x sectors / 2 = disk size in logical blocks

Also calculate the number of logical blocks per track (call this value bpt):

sectors / 2 = bpt

Finally, calculate total number of tracks:

heads x cylinders = number of tracks on the disk

Some commands described below require values expressed in numbers of logical blocks. Others, however, require values expressed in physical sectors. One logical block equals two physical sectors.

Planning the Disk

In planning the size of the disk's slices, consider the applications or users who are likely to use the disk. How many 1024-byte logical blocks is each application or user likely to require? If you have a lot of users who will create a lot of small files, they can share a file system, provided none of them is careless about using up extra space. What's a rough estimate of their total requirements?

Try to put user files in a file system separate from the root file system and other file systems that hold system utilities.

Slices always contain a whole number of tracks. Round off your estimates to the nearest multiple of bpt.

The first and most important slice is the reserved area. Preparing a reserved area that supports a system boot is beyond the scope of this manual, but if you reconfigure a disk that contains an operating system, do not decrease the size of the reserved area. (See "Reconfiguring Old Disks," below.) A reserved area that does not support a system boot needs one track.

The Disk Description File

A disk description file is a text file that gives the disk's physical characteristics and tells where each slice begins. To create the disk description file, do the following:

1. Copy the appropriate prototype description file from **/usr/lib/iv**.

2. Use a text editor to add a disk name and slice information to the copy.

The disk name goes on an existing line. The line is of the form

name name

where

name identifies the name line.

is a tab character, generated by pressing the TAB key or Control-I.

name is the disk name. Only the first six characters are used. If the specified name is less than six characters long, the actual name is padded out to six with blanks.

Add slice information at the end of the description file. Each slice is described by a line that contains the track number for the first track of the slice. The first track number gives the starting track for slice 0, the second for slice 1, and so on. The lower numbered slices always begin with lower numbered tracks. Slice 0, the reserved area, always begins with track 0, but this slice is still listed explicitly.

It is not absolutely necessary that the disk description file have any particular name or be in any particular directory. However, you will find it useful to keep all your disk description files in a special directory, separate from **/usr/lib/iv**, and to name them in a way that indicates their purposes. For some purposes, such as backup, you will require a large number of identically initialized disks, so you will reuse some disk description files.

Example 1

An administrator wants to use diskettes. The description prototype file that describes his disk is **pdl.desc** (this name is for example only). The administrator's input is in boldface, the system's responses in normal type.

```
# cat /usr/lib/iv/pdl.desc
type            FD
name            xxxxxx
cylinders       80
heads           2
sectors         8
```

```
steprate        Ø
$
$
$
```

Sectors is even, so leave it alone. The capacity of the disk is 2 x 8Ø x 8 / 2 = 64Ø logical blocks. There are 8 / 2 = 4 blocks per track (bpt) and 8Ø x 2 = 16Ø total tracks. If the disk doesn't have an operating system, one track is sufficient for the the reserved area (Slice Ø), leaving 159 tracks (636 logical blocks) for data.

The system administrator plans two uses for the diskettes:

- Backup. Each slice 1 will serve as a "tape reel" in backups. The disks only need to be initialized and configured; the backup programs will manage the slice.
- Offline file storage. Users will use these disks to increase their storage. Slice 1 will contain a file system; users will create files on the file system.

A single disk description file will initialize disks for both purposes.

```
# mkdir /usr/lib/iv.work
# cd /usr/lib/iv.work
# cp /usr/lib/iv/pdl.desc pd.offline
# ex pd.offline
"pd.offline", 9 lines, 63 characters
: 1,$ #
   1  type            FD
   2  name            xxxxxx
   3  cylinders       8Ø
   4  heads           2
   5  sectors         8
   6  steprate        Ø
   7  $
   8  $
   9  $
: 2 s/xxxxxx/offline/p
name   offline
: $ append
Ø
1
.
: 1,$ #
   1  type            FD
   2  name            offline
   3  cylinders       8Ø
   4  heads           2
   5  sectors         8
   6  steprate        Ø
   7  $
```

```
   8  $
   9  $
  10  0
  11  1
: xit
"pd.offline", 11 lines, 73 characters
#
```

Note that **pd.offline** will name new disks "offlin": disk names are always 6 characters long, and specified names are right-truncated or padded with blanks to fit.

Running Iv

The **iv** program formats a disk and divides it into slices. This program is used to initialize new disks and rearrange the slice boundaries on old disks.

The **iv** command takes one of the following forms:

/etc/iv /dev/rfp0_t_**0** _file_

/etc/iv -f /dev/rfp0_t_**0** _file_

where

_t_ indicates the disk to be formatted or reconfigured: 0 means the fixed disk; 2 means the disk inserted in the diskette drive.

file is the name of a disk description file.

The second form suppressing formatting, so that slices whose boundaries do not change should not be damaged. However, it is not safe too assume that any data will survive application of **iv** to a disk: always copy the contents of a disk before using **iv** on it.

MAKING AND USING A FILE SYSTEM

Each file system requires the following steps:

- Create the file system with **mkfs.**
- Mount the file system and arrange for its automatic mounting.
- Create a **lost+found** directory for the file system.
- If the file system is on the fixed disk, add it to the operating system checklist.

Creating the File System

Mkfs creates a file system by writing file system structures into a slice. **Labelit** puts a label on the slice.

/etc/mkfs /dev/fpØtp size 1 cylsize
/etc/labelit /dev/fpØtp ldir vname

where

t — indicates the disk that contains the slice: Ø means the fixed disk; 2 means the disk inserted in the diskette drive.

p — is the slice number, in hexadecimal (**a** through **f** stand for "1Ø" through "15"). Do not specify slice Ø (the reserved area).

size — is the number of sectors in the slice. This is the twice the number of logical blocks in the slice.

cylsize — is the number of sectors per cylinder. This is the number of heads, times bpt, times 2.

ldir — is the local name of the directory on which the file system is normally mounted or **root** for the root file system. Examples: a file system normally mounted on **/a** is **a**; a file system normally mounted on **/usr/src** is **src.**

vname — is your name for the disk that holds the file system. Suggested names: **dØ** for the fixed disk, **d2** for the diskette.

Example

The administrator from the previous example wants a file system on a diskette. Cylsize is 2 x 4 x 2 = 16. The administrator inserts a blank disk in the drive enters the following commands. The system administrator's input is in boldface, the computer's responses in normal type.

 # /etc/iv /dev/fpØ2Ø /usr/lib/iv.work/pd.offline
 # /etc/mkfs /dev/fpØ21 1272 1 16

Mounting a File System

Every file system must be mounted in order to be used; this gives the file system a place in the file hierarchy. The **mount** command mounts a file system:

/etc/mount /dev/fp0_dp_ _dir_ _r_

where

_d_ indicates the disk, as in the **mkfs** command.

_p_ indicates the slice, as in the **mkfs** command.

dir is the name of an empty directory. Subsequent references to _dir_ will actually be to the root directory of the newly mounted file system; this gives user normal access to any files on that file system.

This directory can be on the root file system or it can be on another mounted file system. But the directory's file system must be already mounted.

_r_ controls access to the file system. If _r_ is **-r**, the file system is mounted read only: files in the file system can be read, subject to normal permission rules, but cannot be modified, created, or deleted by any user. If _r_ is missing, the file system is mounted read/write: all files in the file system can be read, modified, created, or deleted, subject to normal permission rules.

Without any arguments:

/etc/mount

mount displays a list of currently mounted file systems.

A **umount** undoes a **mount**:

/etc/umount /dev/fp0_dp_

where

_d_ indicates the disk, as in the **mkfs** command.

_p_ indicates the slice, as in the **mkfs** command.

The root file system (the file system in slice 1 of the disk from which operating system was booted) is always accessible without a mount. Do not apply **umount** or **mount** to this file system.

Mounting and unmounting is automatic in some circumstances:

- The operating system normally mounts fixed disk file systems automatically when the operating system goes to multiuser mode. To accomplish this, the system administrator must add commands to the appropriate startup file; see "Using Fixed Disks," below.
- Unmounting is implied by the **dismount** program, which cleanly disconnects a removable disk from the operating system. See "Using Removable Disks," below.
- The **shutdown** program, which takes the system to single-user mode, unmounts all file systems except the root file system. See Section 2.

Creating the Lost+Found Directory

Each file system must have a special directory for use by the file system maintenance program **fsck**. To create this directory:

1. Make sure that the file system is mounted.
2. Make the file system's root directory your working directory.
3. Run the program:

   /etc/mklost+found

Editing the Checklist

If the file system is on the fixed disk, its integrity should be checked whenever the System 6300 is booted. To do this, use a text editor to modify the file **/etc/checklist**. The first line must specify the record special file for the root file system:

/dev/fp001

Each additional file system must be represented by a line that specifies the file system's character special file:

/dev/rfp00*p*

where

p is the partition number in hexadecimal (**a** through **f** stand for "10" through "15").

Make sure that there are no duplicate entries. It is also important that the root file system be listed first.

USING THE FIXED DISK

All fixed disk file systems can and should be mounted automatically when the system is booted. To insure this, put the correct **mount** commands in **/etc/mountable.**

USING REMOVABLE DISKS

After inserting a removable disk in the drive, wait for the disk to reach full speed. If a removable disk contains file systems, issue the following command for each file system:

/etc/mount /dev/fpØt**p /mnt/exch**p

where

t identifies the kind of disk.

p is the slice number in hexadecimal (**a** through **f** stand for "1Ø" through "15").

Users can access the file systems on the removable disk by referring to the subdirectories of **/mnt.**

The system administrator must specify who can access the root directory of a removable disk file system. Since removable disks are relatively small, one of two possibilities is likely:

- One particular user controls the root directory.
- All users are allowed to use the root directory.

To give a particular user control of the root directory, use the **chown** program:

/etc/chown name **/mnt/exch**p

where

name is a user name.

p is the slice number in hexadecimal.

The user can then specify access permissions for the directory. If the user gives only himself write permission, he has the exclusive ability to create files and subdirectories in **/exch**p.

To give all users free access to the root directory, and thus to the file system, use the **chmod** program:

chmod a+rwx /mnt/exchp

where

p is the slice number in hexadecimal.

All users can then create files and subdirectories in **/exchp**.

Before removing the disk from the drive, use **dismount:**

dismount -f

Dismount dismounts the disk's file systems, completes input/output, and tells the operating system kernel that removal of the disk is safe.

If a disk is removed without a complete **dismount**, it is marked as potentially inconsistent. If a potentially inconsistent disk is inserted in the drive, a warning message will appear on the console.

CHECKING FILE SYSTEM INTEGRITY

If the operating system stops running for any reason, it is time to check the integrity of whatever file systems were mounted when the operating system went down. This is especially true after unplanned down times (operating system crash, power outage, accidental reset of the system), but is also mandatory for planned down times. The **fsck** program examines the file system and repairs damage to file system structures.

Routine Checks of Fixed Disk File Systems

The operating system automatically checks fixed disk file systems whenever it is booted. **Fsck** is run with an option that performs most repairs automatically. If file system problems are too difficult for **fsck** to handle automatically, the operating system goes to administrator mode. If this happens, log in as root on the most convenient terminal and run **fsck** manually, as described below.

Routine Checks of Removable Disk File Systems

Check the integrity of a removable disk's file systems

- Whenever the disk sees a week's worth of work.
- Whenever the system warns that the disk has been pulled. This will happen if the disk was in use when the system was stopped suddenly.

The operating system must be in single user state. The following two commands are required:

/etc/umount /dev/fp0t**p**
/etc/fsck -p /dev/fp0t**p**

where

t indicates the kind of disk: 2 means diskette.

p is the slice number, in hexadecimal (a through f stand for "10" through "15").

The first command assures that the file system is unmounted; this is very important.

Fsck should handle most problems automatically. If it cannot, run it again manually.

Running Fsck Manually

To manually check a file system, run **fsck** without the **-p** option:

fsck /dev/fp0t**p**

where

t indicates the kind of disk: 0 means the fixed disk, **2** means the diskette.

p is the slice number, in hexadecimal (a through f stand for "10" through "15"). Do not specify slice 0 (the reserved area)!

This form of the **fsck** command requires your permission before each action.

Repair the normal root file system (**/dev/fp001**) first. This simplifies work on the other file systems. If **fsck** actually modified the root file system, it will reboot the operating system.

If you can't understand **fsck**'s actions, do the following:

1. Familiarize yourself with the following **fsck** description and with Appendix A.

2. Continue running **fsck**, but assume you'll have to run it again. Grant permission only for minor repairs. Assess the condition of the file systems. (Examples of minor repairs: removing a small or unimportant file; linking a file to **lost+found**.

3. For each sick file system, consider how much work would be lost if the file system were thrown away and restored from back up. If this loss is not significant, abandon further work on the file system and restore from back up.

Fsck Description

These are the messages and suggested actions **fsck** displays. Run with the **-p** option, **fsck** takes most of its own suggestions, stopping only when action would mean loss of data.

For a discussion of the terminology employed in this section, see Appendix A.

These messages each report a disk problem and suggest a standard fix. These are the standard suggestions:

(CONTINUE) Results may be invalid. Continue anyway?

(CLEAR) Clear this i-node?

(REMOVE) Remove this directory entry?

(FIX) This value is wrong. Replace it with the right value?

(RECONNECT) We have a good i-node but no directory entry for it. Make a directory entry for it in **lost+found**?

Initialization

CAN NOT SEEK: BLK B **(CONTINUE)**
CAN NOT READ: BLK B **(CONTINUE)**
CAN NOT WRITE: BLK B **(CONTINUE)**
I/O failed on the file system. If you decide to continue, do a second run to confirm the results of the first. Make sure the disk isn't write-protected.

PHASE 1: CHECK BLOCKS AND SIZES

UNKNOWN FILE TYPE I=I **(CLEAR)**
I-node I has an invalid type. If you decide to clear the i-node, its directory entries will be **UNALLOCATED** in Phase 2.

LINK COUNT TABLE OVERFLOW (CONTINUE)
An **fsck** internal table is full. A second **fsck** run will be necessary to confirm the results of the first. This

message will repeat each time **fsck** encounters an allocated i-node whose link count is Ø.

B BAD I=I
Block B on i-node I is bad. You'll get a **BAD/DUP** message in Phase 2 and Phase 4.

EXCESSIVE BAD BLKS I=I (CONTINUE)
I-node I has a large number of bad blocks. If you choose to continue, **fsck** will skip to the next i-node. Run **fsck** again to verify your results.

B DUP I=I
I-node I claims block B, but this block is already on **fsck**'s list of allocated blocks. This will cause a **BAD/DUP** message in Phase 2 and Phase 4. This error invokes Phase 1b. Be careful if you see this error. A good procedure is to note the i-numbers with this error and finish running fsck without changing the file system. Before running fsck again, run **ncheck** to discover the names of the affected files:

/etc/ncheck -i numbers **/dev/**name

where

numbers — is a list of i-numbers. The numbers are separated from each other by spaces.

name — is the same special file name used with **fsck**.

EXCESSIVE DUP BLKS I=I (CONTINUE)
I-node I has a large number of duplicate blocks. If you choose to continue, **fsck** will skip to the next i-node. Run **fsck** again to verify your results.

DUP TABLE OVERFLOW (CONTINUE)
An **fsck** internal table is full. A second **fsck** run will be necessary to confirm the results of the first. This message will repeat each time **fsck** encounters a duplicate block.

POSSIBLE FILE SIZE ERROR I=I
The indicated file has the wrong number of blocks for a file its size. A file size error does not necessarily indicate a real error: it can indicate a file that was written to nonsequentially.

DIRECTORY MISALIGNED I=I
Size of the indicated directory is not a multiple of 16.

PHASE 1B: RESCAN FOR MORE DUPS

B **DUP I=**I
I-node I claims block B, but this block is already on **fsck**'s list of allocated blocks.

PHASE 2: CHECK PATHNAMES

ROOT I-NODE UNALLOCATED. TERMINATING
The root directory of a file system is always linked to i-node 2. If i-node 2 is not allocated, the file system is damaged beyond repair.

ROOT I-NODE NOT DIRECTORY (FIX)
I-node 2 must be a directory.

DUPS/BAD IN ROOT I-NODE (CONTINUE)
Some of the duplicate blocks belong to i-node 2. This will make it very difficult to repair the file system.

I **OUT OF RANGE I=**I **NAME=**F **(REMOVE)**
F, a directory entry, refers to a i-node that doesn't exist.

UNALLOCATED I=I **OWNER=**O **MODE=**M **SIZE=**S **MTIME=**T **DIR=**F **(REMOVE)**
The specified directory entry is a link to an unallocated i-node.

DUP/BAD I=I **OWNER=**O **MODE=**M **SIZE=**S **MTIME=**T **DIR=**F **(REMOVE)**
The specified directory entry is a link to a i-node with bad blocks or blocks duplicated by another file.

BAD BLK B **IN DIR I=**I **OWNER=**O **MODE=**M **SIZE=**S **MTIME=**T
You specified the **-q** option and **fsck** spotted inconsistent data in the specified directory.

PHASE 3: CHECK CONNECTIVITY

UNREF DIR I=I **OWNER=**O **MODE=**M **SIZE=**S **MTIME=**T **(RECONNECT)**
The indicated directory is nonempty and uncorrupted but lacks a directory entry (its former parent has no link to it).

SORRY, NO lost+found DIRECTORY
No files can be reconnected until you replace the missing **lost+found** directory. If you really need to reconnect your unreferenced i-nodes: first, finish this **fsck** run (being careful not to clear any i-nodes!); then recreate the missing **lost+found** directory (see "Creating the Lost+Found Directory," above); and finally run **fsck** on the file system again.

SORRY, NO SPACE IN lost+found DIRECTORY
No more files can be reconnected until you expand the **lost+found** directory. If you really need to reconnect your unreferenced i-nodes: first, finish this **fsck** run (being careful not to clear any i-nodes!); then expand the **lost+found** directory (see "Creating the Lost+Found Directory," above); and finally run **fsck** on the file system again.

DIR I=I1 CONNECTED. PARENT WAS I=I2.
The directory whose i-number is I1 now has a link in **lost+found. Fsck** has made the directory's .. entry refer to **lost+found**; formerly .. referred to i-node I2.

PHASE 4: CHECK REFERENCE COUNTS

UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT)
The indicated ordinary file is nonempty and uncorrupted but lacks a directory entry (the directories that had links to it lost them).

SORRY, NO lost+found DIRECTORY
SORRY, NO SPACE IN lost+found DIRECTORY
See the **lost+found** messages in Phase 3.

(CLEAR)
A chance to abandon the last **UNREF** file without rerunning **fsck.** Be absolutely sure you want to clear this i-node.

LINK COUNT FILE I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST)
The link count for the specified ordinary file is X but Y files actually have links to it.

LINK COUNT DIR I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST)
The link count for the directory is X but Y files actually have links to it.

UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)
UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)
Ordinarily, unreferenced and empty files and directories silently disappear. If the -n option is specified, this prompt appears for empty files and directories.

BAD/DUP DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)
BAD/DUP FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)
The specified directory or file is unreferenced and has bad or duplicate blocks.

FREE I-NODE COUNT WRONG IN SUPERBLK (FIX)
Fsck's count of free i-nodes doesn't match the count in the superblock.

PHASE 5: CHECK FREE LIST

EXCESSIVE BAD BLKS IN FREE LIST (CONTINUE)
EXCESSIVE DUP BLKS IN FREE LIST (CONTINUE)
This is your last chance to avoid reconstructing the free list.

BAD FREEBLK COUNT
X BAD BLKS IN FREE LIST
X DUP BLKS IN FREE LIST
X BLKS MISSING
Final notes on the dire state of the free list. Any of these will invoke the **BAD FREE LIST** message.

FREE BLK COUNT WRONG IN SUPERBLOCK (FIX)
Fsck's count of free blocks does not match the value in the superblock.

BAD FREE LIST (SALVAGE)
If the file system is otherwise all right, it's always a good idea to salvage the free list.

Rebooting the System

The kernel can undo all your painful repair work by writing out its copies of file system tables. Give the kernel no chance to do I/O to the file systems until you return to multiuser state or halt the processor:

- Unmount all file systems (except the root file system, of course). Keep them unmounted.
- Under no circumstances run **sync** from the time you start repairing the file systems to the time you switch to multiuser state or halt the processor.

SECTION 5: USER SUPPORT

ADDING USERS

The following actions give a new user basic access to the the system:

1. Assign the user a unique log-in name.
2. Choose a file system and home directory to hold the user's file.
3. Create an entry for the user in the password file.
4. Create the user's home directory.

Log-In Names

The log-in name uniquely identifies the user. The log-in name must be one to eight characters long and consist of letters or digits. If the name has letters, at least one of them must be lower case. Two users must not have the same log-in name.

Choosing a File System and Home Directory

Decide which file system will hold the user's permanent files. If possible, reserve one or more file systems for non-administrative users. Divide the file systems between users based on your estimate of the users' storage needs.

Name user home directories in a way that is convenient for you. One good system follows two conventions:

- The user's home directory is in the root directory of the user's file system.
- The simple name of the home directory is the same as the user's log-in name.

Decide on the name of the user's home directory, but don't create the actual directory yet. This will be easier to do once you have created the user's password file entry.

Example 1

Suppose four new users choose log-in names "john," "dick," "gwen," and "jack." The system administrator decides that there is room for them on the file system whose special file name is **/dev/fp003.** This file system is normally mounted on /a. The new users' home directories will be **/a/john, /a/dick, /a/gwen,** and **/a/jack.**

The Password File

Each user requires an entry in the Password File, **/etc/password.** This file is read by **login** every time someone tries to log in. To add a new user to the password file:

- Use a text editor to add the user's entry to the file.
- Use the **passwd** program to assign a password to the new user.

Each entry in the password file is a line of the form

name:pass:uid:gid:unused:home:shell

where

name	is the user's log-in name.
pass	is user's encrypted password. Leave this field blank; it will be filled in when you run the **passwd** program.
uid	is the user's numeric user ID. This must be a decimal number greater than or equal to 100 and unique for each user.
gid	is the user's initial numeric group ID. To implement groups, see group(4) in the Series 6000 Operating System Reference Manual. If the user is not associated with any group, set this field to 100.
unused	is a field without any standard use. It is often holds the user's name and office location.
home	is the name of the user's home directory.

shell — is the full path name (not the command name) of the user's shell, the program that is executed when the user logs in. If this field is empty, **login** uses the Bourne shell, **/bin/sh.**

To specify a password for the new user, run **passwd.** The form of the command is

passwd name

where name is the user's log-in name. **Passwd** prompts for a password. The password does not appear on the screen. To prevent error, **passwd** makes you type the password twice. If you supply a short password, **passwd** demands a longer one, but gives in if you are persistent.

If you fail to supply an encrypted password (leaving that field of the password file entry blank) the user requires no password to log in.

Example 2

The system administrator creates password file entries for "john," "dick," "gwen," and "jack." Each of these users uses the Bourne shell. The system administrator also invents a user that exists only to allow people to run **sync** without logging in. The administrator's input is in boldface, the computer's response is in normal type, and ### indicates unechoed input.

```
# ed /etc/passwd
5328
$p
frank:UCOW7.pjZUBcw:115:100::/a/frank:
a
john::116:100::/b/john:
dick::117:100::/b/dick:
gwen::118:100::/b/gwen:
jack::119:100::/b/jack:
sync::120:100::/:/bin/sync
.
w
5460
q
# passwd john
new password: ###
retype new password: ###
# passwd dick
new password: ###
retype new password: ###
# passwd gwen
new password: ###
retype new password: ###
```

```
# passwd jack
new password: ###
retype new password: ###
#
```

The administrator has kept the password file ordered by numerical user ID, so it's only necessary to examine the last entry to determine the next numerical user ID.

The "sync" user requires no password and no home directory of its own.

User Home Directory

The new user's home directory requires the following steps.

1. Create the directory.
2. Give the user ownership of the directory.
3. Give the user's group group ownership of the directory.
4. Set the protection mode of the directory.

Use **mkdir** to create the home directory. The command has the for

mkdir dir

where dir is the home directory name. Actually, any number of directory names are permitted.

Use **chown** to give the user ownership of the home directory. The command has the form

chown name dir

where

name is the user's log-in name.

dir is the user's home directory.

Use **chgrp** to give the user's group group ownership of the home directory. The command has the form

chgrp group dir

where

group is the name of the group or the numerical group ID.

dir is the user's home directory.

Use **chmod** to set the protection mode of the user's home directory. The command has the form

chmod mode dir

where

mode is a protection mode. **755** gives the owner all access to the directory, and gives other users read-only access. **700** gives the owner all access to the directory and gives other users no access at all. For other modes, see chmod(1) in the Series 6000 Operating System Reference Manual.

dir is the user's home directory.

Each of these commands can do multiple files or directories.

Example 3

The system administrator provides home directories for "john," "dick," "gwen," and "jack." The administrator makes each directory with all access for the owner and read-only access for other users. The administrator's input is in boldface, the computer's response in normal type.

```
# mkdir /b/john /b/dick /b/gwen /b/jack
# chown john /b/john
# chown dick /b/dick
# chown gwen /b/gwen
# chown jack /b/jack
# chgrp 100 /b/john /b/dick /b/gwen /b/jack
# chmod 755 /b/john /b/dick /b/gwen /b/jack
#
```

BARRING AND DELETING USERS

This section describes procedures for denying users access to the system. The subsection "Barring a User" tells how to deny access when the user may not be removed permanently. The subsection "Permanently Removing a User" tells how to completely undo the steps that gave the user access and remove the user's files.

Barring a User

To bar a user without permanently removing him or her, invalidate the user's password file entry. One way to do this is to insert a **%** at the beginning of the entry's encrypted password; use a text editor to do this.

When a user's password file entry is invalid, any attempt by that user to log in is rejected as "incorrect."

To restore the user, remove the **%**.

Example 4

The system adminstrator bars "jack." The administrator's input is in **bold**, the computer's responses in normal type.

```
# ed /etc/passwd
7454
/jack/
jack:wcBUZjp.7WOCU:119:100::/b/jack:
s/:/:%/p
jack:%wcBUZjp.7WOCU:119:100::/b/jack:
w
7455
q
#
```

Permanently Removing a User

It is a good idea to postpone permenant removal of a user until after regular file backups. If this is inconvenient, consider using the temporary procedure, above, until the next regular backup.

To remove a user from the system permenantly:

1. Remove the user's password file entry.

2. Remove the user's files.

Use a text editor to remove the password file entry.

Rm will remove the user's home directory and all the files it contains. The command takes the form

rm -fr *dir*

where

dir is the user's home directory.

CAUTION

> The above form of the **rm** command can remove a large number of directories quickly. Note that the **rm** command does not announce the files it is removing. Use the above command carefully and precisely.

If a user's file outlasts the user's password file entry, an **ls -l** on the file produces the former user's numeric user ID.

The former user may leave files in places other than his home directory. The following command does a comprehensive search for the former user's files, but takes a lot of time to execute:

find / -user x **-print**

where x is the user's log-in name or numeric user ID. If the user's password file entry is gone the log-in name will not work but the numeric user ID will.

Example 5

The system administrator removes "jack" and that user's files.

```
# ed /etc/passwd
7530
/jack/
jack:%wcBUZjp.7WOCU:119:100::/b/jack:
d
w
7490
q
# rm -rf /b/jack
#
```

MOVING USERS

If space runs short on a file system, you may need to move a user to a new file system. This requires the following steps.

1. Inform the user of his or her new home directory name.
2. Copy the user's files to their new location.
3. Update the user's password file entry.
4. Delete the user's old files.

Use the same system for assigning moved home directory names that you use for assigning new home directory names.

The following command will create the file copies. The copies will have the same modification dates as the originals; this is desirable if the user uses **make**.

(cd old **; find . -depth -print** | **cpio -pdm** new **)**

where

old is the full name of the old directory.

new is the full name of the new directory.

To remove the old user files, use **rm**:

rm -rf old

where

old is the full name of the old directory.

Example 6

The system administrator from the previous examples wants to move "frank" from the root file system to file system mounted on /a. The system administrator's input is in bold; the computer's responses in normal type.

```
# ed /etc/passwd
7000
/frank/
frank:UCOW7.pjZUBcw:115:100::/frank:
s/\//\/a/p
frank:UCOW7.pjZUBcw:115:100::/a/frank:
w
7000
q
# (cd /frank; find . -depth -print | cpio -pdl /a/frank)
# rm -r /frank
#
```

SECTION 6: BACKUPS AND RESTORES

Offline backup protects fixed disk files from the unexpected. Backup provides copies of files and file systems against accident, carelessness, and technical mishap.

Backups require three kinds of chores:

- Scheduling backups.
- Doing backups.
- Restoring files and file systems from backups.

SCHEDULING BACKUPS

The following sample schedule has the basic features of a good backup schedule.

- Permanent total. Every fourth Friday, each file system is completely copied. The copies are saved permanently.
- Temporary total. Every Friday, except on days when a permanent volume backup is done, each file system is completely copied. The copies are saved for four weeks.
- Incremental. Every working day, except on days when volume backups are done, all files created or modified since the last volume backup are copied. The copies are saved for a week.

Backups occur at the end of the working day.

Some of the features of this schedule are arbitrary, some are not. Every four weeks may be too often for you to make permanent backups; but if you increase the time between permanent total backups, make the same increase in the time you keep temporary total backups. Total backups need not occur on Fridays, but should occur at the same time each week; backups need not occur at the end of the working day, but the time they do occur should not change from day to day.

The most important feature of this schedule is that it does not permit the loss of more than a day's work due to the complete loss of the fixed disk's files. It also protects files against accidental removal: the longer a file is left on the fixed disk, the harder it is to lose it permanently.

DOING BACKUPS

The following steps are required to backup a file system:

1. Take the system to single-user mode, as described in Section 2.
2. Prepare enough data diskettes using the procedures in Section 4. Each diskette should have a minimal slice Ø and a slice 1 that takes up the remainder of the diskette. Do not create a file system in slice 1.
3. Use the appropriate program to do the actual backup.
4. Print out a log of the files backed up.
5. If this is a total backup, register the time for the benefit of this week's incremental backups.

There are two distinct backup procedures:

- The total backup. Each fixed disk file system is separately copied onto a set of diskettes.
- An incremental backup. A list of files modified since the last total backup is prepared and each file on the list is copied to a offline archive.

Total Backups

The **labelit** and **volcopy** commands accomplish a total backup of a single file system. To do a total backup, insert the first backup diskette in the drive and execute the following command:

/etc/labelit /dev/rmtd ldir **backup -n**
/etc/volcopy -a ldir **/dev/rfp ØØ**s vname **/dev/rmt**d **backup**

where

d — indicates the drive that holds the backup volume: 1 for diskette.

ldir — is the local name of the directory on which the file system is normally mounted or **root** for the root file

system. Examples: a file system normally mounted on /a is a; a file system normally mounted on /usr/src is src.

s is a slice number of the file system that is to be backed up. The number is hexadecimal (a through f stand for "10" through "15").

vname is your name for the disk that holds the file system.

To generate a log of files backed up, remount the file system and use **ff** with **lpr**:

```
/etc/mount /dev/fp00s dir
/etc/ff -p dir -s -u /dev/fp00s | lpr
```

where

dir is the name of the directory on which the file system is normally mounted.

s is the hexadecimal slice number.

The log will appear shortly on the system printer. Keep it in a safe place.

In this and other backup commands, the root file system is considered to be mounted on the / directory.

A file modification time can register the time of the total backup.

```
> /TOTAL
```

Incremental Backups

An incremental backup copies some files from all fixed disk file systems. This is different from a total backup, which copies all files from specified file systems. The following steps accomplish an incremental backup.

1. Remount the mountable file systems in their normal place.
2. Generate a list of file modified since the last total backup.
3. Copy every file in the list to an offline archive.
4. Print out the list.

The mountable file systems were unmounted when you took the operating system to signle-user mode. To remount them, execute the startup mounting procedure:

sh /etc/mountable

Use the **find** command to generate a list of recently modified files:

find / -newer /TOTAL -print | sort > /INC_d_

where

_d_ is the number of days since the total backup.

Use the **cpio** command to copy the recently modified files to the archive. Insert the first diskette in the drive and do:

cpio -ocvB < /INC_d_ **> /dev/rmt**_t_

where

_d_ is the number of days since the total backup.

_t_ is the type of backup medium; 1 for diskette.

When **cpio** uses up the diskette, it will print the follow message:

Error 6: Can't write output
If you want to on, type device/file name when ready

The "can't write output" error is normal. Remove the diskette and insert the next one in the set. (This will produce a "possibly inconsistent" message, which you safely ignore.) Now type the diskette device name:

/dev/rmt_t_

where

_t_ is the type of backup medium; 1 for diskette.

You will continue to get Error 6 messages until all the files are copied. Switch diskettes and retype the diskette device name each time the message appears. Follow this procedure carefully; if you accidentally terminate **cpio**, run it again and start over with the first diskette.

To print out the log of files backed up, do:

xargs ls -ld < /INC_d_ **| lpr**

where

_d_ is the number of days since the last total backup.

Unless you plan to go directly back to multiuser mode, be sure to unmount the file systems again. The surest way to do this is to run **halt**:

```
/etc/halt
```

The Backup Log

The log printout is different for a total backup and an incremental backup. The total backup log lists four data on each file backed up.

- The full file name. The first part of this name is the name of the directory on which the file's file system is mounted.
- The file's i-number. The i-number is unique for each file on a file system, but not for each file name. If two file names list the same i-number, they are two links to one file.
- The file's size, in bytes.
- The login name of the file's owner.

The incremental log uses the format of the **ls** command. See **ls**(1) in the Series 6000 Operating System Reference Manual.

RESTORES

A mishap can destroy an entire file system or just a few files. Restoring an entire file system requires copying the file system from the last total backup, then copying each of the subsequent incremental backups. Restoring specific files simply means copying those files from the latest backups that have them.

Restoring an Entire File System

The following steps completely restore a file system:

1. Take the operating system to single-user mode, as described in Section 2.
2. Copy the file system from the total backup.
3. Mount the file system on its normal mount directory.

4. Retrieve the file system's files from the incremental backups.

To restore a total backup, do:

/etc/volcopy ldir **-a /dev/rmt1 backup /dev/rfp00**s vname

where

ldir — is the local name of the directory on which the file system is normally mounted or **root** for the root file system. Examples: a file system normally mounted on **/a** is **a**; a file system normally mounted on **/usr/src** is **src**.

s — is the slice number of the file system that is to be backed up. The number is hexadecimal (a through f stand for "10" through "15").

vname — is your name for the disk that holds the file system.

Note the reversal in parameters from the command that backed up the file system.

Restoring a file system this way completely rewrites file system data structures, using the version on the backup disks. If the file system was sick before the restore, **volcopy** cures it of its current problems, but restores any problems it had at the time of the total backup.

To completely restore the incrementally backed up files, restore each incremental backup for the file system. Restore the oldest backup first, but do not restore any backup made before the last total backup.

This procedure restores an incremental backup. Insert the first disk in the incremental set in the drive and type

cpio -iBcduvm 'dir**/*' < /dev/rmt**t

where

dir — is the name of the directory on which the file system is normally mounted.

t — is the type of backup medium; 1 for diskette.

When **cpio** reads through the diskette, it will print the follow message:

```
Error 6: Can't read input
If you want to on, type device/file name when ready
```

The "can't read input" error is normal. Remove the diskette and insert the next one in the set. (This will produce a "possibly inconsistent" message, which you safely ignore.) Now type the diskette device name:

/dev/rmt_t_

where

_t_ is the type of backup medium; 1 for diskette.

You will continue to get Error 6 messages until all the files are copied. Switch diskettes and retype the diskette device name each time the message appears. Follow this procedure carefully; if you accidentally terminate **cpio**, run it again and start over with the first diskette.

Restoring Specific Files

Backup disks made with **volcopy** require a different restoration procedure than backup disks made with **cpio**. Having the system in single user mode is not absolutely necessary, but will avoid collisions with users. In any case, the file systems must be mounted; in single user mode the simplest way to make sure that all file systems are mounted is to type:

sh /etc/mnttable

The **frec** command restores individual files from backup disks made by **volcopy**.

/etc/frec /dev/rmt1 _i_**:**_name_ **...**

where

_i_ is the i-number for the file.

name is the name the file will have when it is restored. This is normally the same name it had when it was backed up, but need not be.

... indicate additional files for recovery. Each argument takes the _i_**:**_name_ form and is separated from other parameters by spaces.

Frec will prompt you to insert the backup disks.

Do all recoveries from a single set of backup disks with one **frec** run. If a large number of files are to be recovered, use this procedure:

1. Use a text editor to create a file that lists all the files to be recovered. Each line in the file must be of the form

   i:name

   where i and name mean the same thing they do in a **frec** parameter.

2. Run **frec**:

   /etc/frec -f file

   where file is the name of the file created in step 1.

If a missing file or directory to be restored was in a directory that is also missing, restore the parent directory also. If you fail to do this, **frec** will recreate the missing directory, but it will be a new directory and may not have the same ownership and modes as the original.

Restoring specific files from incremental backup is very similar to restoring an entire file system from incremental backup. Insert the first disk in the set in the drive and type:

cpio -iBcduvm list **< /dev/rmt**t

where

list is a list of files to be recovered.

t indicates the backup medium; **1** stands for diskette.

If you need to recover a large number of specific files, create a text file with all the file names in it, and use this form of the **cpio** command instead of the one above.

cpio -iBcduvm `cat** file**` < /dev/rmtt

where

file is the file that contains a list of files to be recovered.

t indicates the backup medium; **1** stands for diskette.

Note that the second form of the **cpio** command uses accents grave. Do not confuse them with single quotes (').

As with recovering the entire incremental backup, recovering specific files will require intervention each time cpio reads through a diskette. See above.

USER'S COMMENTS

System 6300 Administrator's Guide
S6000-50-1A

HELP!

Help us help you! Please take the time to complete this form and send it to us. If you do, you may see some of your own contributions in the next manual you obtain from us.

- Does this manual provide the information you need? ☐Yes ☐No
  – What is missing?

- Is the manual accurate? ☐Yes ☐No
  – What is incorrect? (Be specific.)

- Is the manual written clearly? ☐Yes ☐No
  – What is unclear?

- What other comments can you make about this manual?

- What do you like about this manual?

- On a scale of 1 to 10, how do you rate this manual? Low ⊢+-+-+-+-+-+-+-+-+-⊣ High
  1 2 3 4 5 6 7 8 9 10
- Was this manual difficult to obtain? ☐Yes ☐No

Please include your name and address if you would like a reply.

Name _____
Company _____
Address _____
_____

No postage required if mailed within the USA.

- What is your occupation?

☐ Programmer
☐ Systems Analyst
☐ Engineer
☐ Operator
☐ Instructor
☐ Student
☐ Manager
☐ Customer Engineer
☐ Other _____

- How do you use this manual?

☐ Reference Manual
☐ In a Class
☐ Self Study
☐ Introduction to the Subject
☐ Introduction to the System
☐ Other _____

fold fold

FIRST CLASS
Permit No. 194
Cupertino, California

BUSINESS REPLY MAIL
No Postage Necessary if Mailed in the United States

Postage will be Paid by . . .

MOTOROLA, INC.
10700 North De Anza Blvd.
Cupertino, CA 95014

Attention: Technical Services, MS 42-1C8

Cut Along ...ere

fold fold

Staple Here

APPENDIX A: FILE SYSTEM CONCEPTS

ACCESS TO PERIPHERAL DEVICES

The opeating system provides a standard way for programs to use peripheral devices. Each special file represents a particular way to access a particular peripheral. A special file appears on the file system (by convention, in **/dev**) and ordinary input/output operations on special files have standard meanings standard for the peripheral.

Special files are either block or character. Block special files identify kernel routines that are most efficient with input/output operations precisely 1Ø24 bytes long. Character special files identify kernel routines that don't prefer any particular size operation. Some kinds of peripherals are represented by both block special files and character special files.

The System 63ØØ operating system is normally set up with 32 special files for each disk drive, providing a block special file and a character special file for each possible slice. The name of a disk's special files takes the form

/dev/r**fpØ**t**p**

where

r is missing on block special files and is **r** (for "raw") on character special files.

t indicates the particular drive: **Ø** for fixed disk, **1** for cartridge disk, **2** for diskette.

p is the slice number in hexadecimal.

If a disk has fewer than 16 slices, it is an error to use the special files for the nonexistent slices.

SECTORS AND BLOCKS

A block is the basic unit of disk input/output. There are two kinds of block:

- Physical sector. This is a physical entity 512 bytes long. A disk drive's basic access to the disk reads or writes a physical sector.

- Logical block. This is a conceptual entity 1024 bytes long. An input/output operation involves 1 or more (never a fraction) logical blocks. Using double-size logical blocks improves performance.

The utilities that initialize the disk, create the file systems, and report on disk sizes consider blocks synonymous with physical sectors. But most programs, including **fsck**, consider the basic unit to be the logical block. In the remainder of this appendix, a "block" is a logical block.

DIRECTORIES

A directory is simply a file that only the operating system kernel can write to. Each directory entry consists of the file name and the file's i-number. A file can have more than one directory entry (link). The number of directory entries that refer to a file is that file's link count.

FILE SYSTEM FORMAT

File system is a storage area (normally a disk slice) with the following structures.

- A block reserved for use in booting the operating system.

- The super block, containing data structures that describe the file system.

- The i-list. This is a sequence of records, called i-nodes, that describe the operating system files. The size of the i-list is fixed when the file system is created. Each i-node has an i-number that gives the i-node's place in the i-list. All file status information is in the i-node, as are the direct and indirect pointers to the file's data blocks.

- The free list. This is a linked list of blocks not used by any file. Each element of the free list is a block that contains pointers to 50 additional blocks.

The program that creates these structures also creates a directory that is the first file on the file system. This directory is the root of the file system.

Two structures in an i-node are important to the administrator: the link count and the disk addresses.

The link count is an integer value. It is Ø when the i-node is not in use. Creating a file sets the link count to 1. Each additional directory entry (link) for the file increments the link count; each removal of a directory entry decrements the link count. If the link count returns to Ø, the file's blocks are returned to the free list -- the file is removed.

There are 13 disk address in the i-node. The first 1Ø point to the first 1Ø blocks of the file (the direct blocks). If the file is more than 1Ø blocks long, the 11th address points to a block that has pointers to the next 256 blocks of the file (the indirect blocks). If the file is more than 266 blocks long, the 12th address points to a block that points to up to 256 blocks containing pointers to the next 65,536 blocks of the file (the double-indirect blocks). If 65,8Ø2 blocks isn't enough, the 13th address provides access to triple-indirect blocks.

These data structures can become inconsistent through incomplete input/output operations, usually those caused by a power failure or through halting the system while the operating system is running full tilt. One of the administrator's jobs is to repair file system data structures using the maintenance programs in described in Section 4.

A mount places a file system on the file system hierarchy. A mount specifies an empty directory and the special file that holds the mountable file system. A mount tells the operating system that any reference to the specified directory is really a reference to the root directory of the file system. The directory on which a file system is mounted itself be on a mounted file system, but naturally the parent file system must be mounted first.

The root file system (the file system whose root is /) is, in effect, always mounted. It is the only file system that has no parent file system.

The term file system actually has two uses, both of them the same on the operating system as on UNIX System V. The documents refer to a file system both as the organization imposed on a single slice ("the file system mounted on /a") and the whole disk hierarchy of files ("the operating system file system"). Context should make clear which is meant.

CAUSES OF FILE SYSTEM CORRUPTION

File system corruption is caused by incomplete or garbled input/output instructions. That can be the result of any of the following:

- Improper shutdown. In particular, all input/output must be complete before the processor is halted. To assure completeness of input/output, kill all user processes and perform two **syncs**. All these procedures are contained in the shell script **/etc/shutdown**.

- Use of a corrupt file system. This causes further errors because of the incorrect file system structures.

- Hardware failure.

FSCK AND THE FILE SYSTEM

Fsck detects errors in three areas:

- The superblock.

- The i-nodes.

- Directory data.

Fsck checks the following in the superblock:

- File system size and i-list size. The file system must be bigger than the superblock plus the i-list. There must not be more than 65,534 i-nodes

  Fsck relies heavily on these two data. Except to check that they are reasonable values, there is no way to confirm their correctness. All other checks depend on the correctness of the file system and i-list sizes.

- Free block list. The first block in the list is in the superblock. Each block in this list contains pointers to additional free blocks. Each block's count of pointed-to blocks must not be less than Ø or greater than 5Ø. Each block pointer must not point past the end of the file system or before the first data block. No block in the free list can be in **fsck's** list of blocks claimed by the i-nodes.

  If **fsck** finds errors in the free list, or if it can't account for every block in the file system, it will ask for permission to reconstruct the free list. The new free list will include all blocks not claimed by any i-node. In the absence of any other serious errors, rebuilding the free list is always safe.

- Free block count. If this does not agree with the actual number of free blocks, **fsck** asks permission to reset the count.

- Free i-node count. If this count is not the same as the size of the i-list minus the number of i-nodes in use, **fsck** asks for permission to reset the count.

Fsck checks the following fields in each i-node:

- Format and type. These fields specify the kind of file (ordinary, directory, block special, character special) and the i-node status (allocated or unallocated). Invalid values indicate that bad data has been written into the i-list. **Fsck** will prompt for permission to clear the i-node; this is always unavoidable.

- Link count. This value must equal the number of directories that actually list the i-node. An inconsistency here indicates a failure to update a directory or the i-node; this is always a minor error.

  If the i-node's link count and the number of links are nonequal and both are nonzero, **fsck** asks permission to correct the i-node link count.

  If the i-node link count is nonzero and the actually link count is zero, **fsck** asks permission to provide a link in the file system's **lost+found** directory.

- Duplicate blocks. These are blocks claimed by more than one i-node. **Fsck** spots duplicate blocks as it builds its list of allocated blocks; this condition requires a second pass of the i-list to find the first i-node. Then **fsck** tries to suggest which i-node should be cleared; usually this is the one with the earlier modify time.

  A large number of duplicate blocks probably indicates that the operating system failed to physically write out a block of pointers to indirect blocks. **Fsck** asks for permission to clear both i-nodes.

- Bad blocks. These are blocks that cannot be found because their addresses are invalid.

  If an i-node has a large number of bad blocks, the operating system probably failed to write out a block of pointers to indirect blocks. Fsck asks for permission to clear the i-node.

- File size. Two kinds of errors can appear here: block allocation consistency and proper directory size.

  Fsck computes the number of blocks required to accommodate a file of the indicated size. If this value doesn't match the number of block the file actual has allocated, **fsck** prints a warning. Note that this condition may be the result of a program seeking past the

end of a file before writing to the file, a perfectly valid action.

If the file is a directory, the file size should be a multiple of 16. If it is not, **fsck** prints a warning but takes no action.

Fsck looks for the following error in directory data:

- Reference to unallocated i-nodes. This probably is the result of the operating system's failure to write out a modified i-node. **Fsck** request permission to remove the directory entry.

- Invalid i-number. This probably is the result of bad data output to the directory. **Fsck** requests permission to remove the directory entry.

- Incorrect . and .. entries: . must be the first entry in the directory and have an i-number equal to the i-number for the directory itself; .. must be the second entry in the directory and be a link to the directory's parent directory. If these entries are incorrect, **fsck** asks for permission to correct them.

APPENDIX B: INIT AND GETTY

1. Introduction

In the UNIX* system environment, the initial process spawning is controlled and overseen by the first process forked by the UNIX operating system as it comes up at boot time. This process is known as *init*. One of the major jobs of *init* is to fork processes which will become the *getty-login-sh* sequence. This sequence of processes allows users to *login* and takes care of setting up the initial conditions on the outgoing terminal lines so that the speed and the other terminal related states are correct. *Init* and these other processes also keep an accounting file **/etc/wtmp** that is available to processes on the system. With these files it is possible to determine the state of each process that *init* has spawned, and if it is a terminal line, who the current user is. One program in particular, *who*(1), provides a means of examining these files.

This document describes the capabilities of each program used in this new implementation, the databases involved, and how to create and maintain these databases. In addition, the debugging features designed into both *init* and *getty* are described in the event remedial action is required or modifications are attempted.

2. Init

Init is driven by a database, its previous internal level, its current internal level, and events which cause it to wake up.

2.1 The Database: /etc/inittab

Init's database, kept in the file **/etc/inittab**, consists of any number of separate entries, each with the form:

id:level:type:process

id — The *id* is a one to four letter identifier which is used by *init* internally to label entries in its process table. It is also placed in the dynamic record file, **/etc/utmp**, and the history file, **/etc/wtmp**. The *id* should be unique.

level — The *level* specifies at which levels *init* should be concerned with this entry. *Level* is a string of characters consisting of [0-6a-c]. Anytime that *init*'s internal level matches a level specified by *level*, this entry is *active*. If *init*'s internal level does not match any of the levels specified, then *init* makes certain that the process is not running. If the level field is empty it is equivalent to the string "0123456".

type — The *type* specifies some further condition required for or by the execution of an entry.

- off — The entry is not to run even if the levels match.
- once — The entry is to be run only if *init* is entering a level. This means if *init* has been awakened by powerfail or because a child died this entry will not be activated. Only when a user signal requests a change of *init*'s internal state to a state which is different from its current state, and this new state is one in which this entry should be active, will this entry be activated.

---

* UNIX is a Trademark of Bell Telephone Laboratories, Incorporated.

wait — *Wait* has all the characteristics of *once*, plus it causes *init* to wait until the process spawned dies before reading anymore entries from its database. This allows for initialization actions to be performed and completed before allowing other processes which might be affected to start running. It is common in the OSS environment for shared memory segments to be initialized this way and semaphores to be conditioned.

respawn — *Respawn* requests that this entry continue to run as long as *init* is running in a level which is in this entry's *level* field. Most processes spawned by *init* fall into this category. All *getty* processes are marked as *respawn*. Whenever *init* detects the death of a process that was marked *respawn*, it spawns a new process to take its place.

boot — *Boot* entries have the execution behavior of *once* entries. They are started only when *init* is switching to a numeric run state for the first time. Most commonly *boot* entries have an empty *level* string, meaning that no matter which level *init* switches to the first time, the *boot* entry will be run. Should there be a more specific *level* string, for example "01", then the *boot* entry would only be run if *init* switched to either the 0 or 1 run state as its first numeric level.

bootwait — *Bootwait* entries have the execution behavior of *wait* entries and they, like *boot* entries, are only run as *init* switches to a numeric level for the first time.

power — *Power* entries act like *once* entries and are activated if *init* receives a SIGPWR signal (19) and is in a state which matches the active states for the entry.

powerwait — *Powerwait* entries act like *wait* entries and are activated if *init* receives a SIGPWR signal and is in a state which matches the active states for the entry.

initdefault — *Initdefault* is a non-standard entry in that it does not specify some process to be spawned. Instead it only specifies which level *init* is to go to initially when it is coming up at boot time. This allows the system to be rebooted without an operator having to make entries at the system console if so desired. If there is no *initdefault* entry, then *init* will ask at the system console, **/dev/syscon,** for the initial run state. In addition to specifying the numbered states, the single-user state [s] may also be specified.

process — The *process* field is the action that *init* will ask a *sh* to perform whenever the entry is activated. The string in the **process** field is given a prefix of "exec " so that each entry will only generate one process initially. *Init* then forks and execs

```
sh -c "exec process"
```

This means that the **process** string can take full advantage of all **sh** syntax. The only peculiarities arise from the string "exec ", which was prefixed to the string, and because initially there is no standard input, output, or error output. The addition of "exec " to the string means that if the user wants to have a single entry generate more than one process, for example making a list of the people on the system at the time of a powerfail and mailing it to **root** by the command "who | mail root", it would have to be put in as

```
pf::powerwait:sh -c "who | mail root"
```

to work. If it was put in simply as "who | mail root", it would be executed as "exec who | mail root", and only the *who* process would be created before the *sh* disappeared. The

lack of standard input and output channels must be addressed by explicitly specifying them. An example is the **blog** program that many OSS's run as a *bootwait* entry as the system comes up. Since it requires the operator to supply input, it appears as

bl::bootwait:/etc/blog </dev/syscon >/dev/syscon 2>&1

in **/etc/inittab**.

2.2 Levels

A level is one of seven numeric levels, denoted 0, 1, 2, 3, 4, 5, or 6, three temporary levels, denoted a, b, or c, or the single-user level, s. Normally *init* runs in a numeric level. Precisely how a particular level is used depends entirely on the database and the system administrator. The temporary levels allow certain entries to be started on demand without affecting any processes that were started at a particular level. The temporary levels immediately revert to the previous numeric level once all entries in the database have been scanned to see if they should be started at the temporary level. When an entry is started by a switch to a temporary level, it becomes independent of future level changes by *init*, except a change to the single-user level. The only way to kill a process that was started as a respawnable demand process, without going to the single-user level, is to modify the database, declaring the entry to be *off*.

The single-user level is the one level independent of the database. For this reason it is not a level in the normal sense. In the single-user level *init* spawns off a *su* process on the system console, and that is the only process that it maintains while at the single-user level. The single-user level can be entered at two different places in *init*. If it is entered at boot time it allows the operator to look over the file systems without having *init* attempt to do any file I/O, which might cause further problems. *Init* will not attempt to recreate **/etc/utmp** or access **/etc/wtmp** until after it has left this initial single-user level. If the single-user level is entered at any other time, *init* does do the bookkeepping in the record files.

The system administrator requests *init* to change levels by running a secondary copy of *init* itself. **/etc/init** is linked to **/bin/telinit**, and it is usually through the *telinit* name that this is accomplished. *Init* can only be run by root or a privileged group. Whenever *init* starts running and finds that its process id is not 1, it assumes that it is a user initiated copy, which is supposed to send a signal to the real *init*. The usage is:

telinit [0123456sSqQabc]

and the single character argument specifies the signal to be sent to *init*. If the request is to switch to the single-user level, 'S' or 's', then *init* also relinks **/dev/syscon** to the terminal originating the request so that it becomes the virtual system console, thus insuring that future messages from *init* will be directed to the terminal where the operator is located. When it does this relinking it also sends a message to **/dev/systty**, saying that the console is being relinked to some other terminal so that there is a record of the fact at the physical system console.

2.3 Waking Events

There are four events which will wake *init*: boot, a powerfail, death of a child process, or a user signal.

boot — *Init* operates in the **boot** state until it has entered a numeric state for the first time. It is not possible for *init* to reenter the boot state a second time. Commands labeled *boot* and *bootwait* are executed when changing to a numeric state for the first time, if the levels match.

powerfail — Any time power fails, the operating system sends a SIGPWR signal to all processes. *Init* will execute commands with types of *power* and *powerfail*.

child death — Any time a child process of *init* dies, *init* receives a SIGCLD signal (18). The dead child process may be one of two types, a direct decendent of *init*, or a process whose own parent process died before it did. The parent of a process automatically

becomes *init*, if its real parent should die before it does. *Init* determines immediately if the defunct process was one of its own children or an *orphan*. If it was one of its own, it performs the necessary bookkeepping on its internal process table to note that the process died. If *init* was busy at the time it received the SIGCLD signal, it then returns to complete whatever action it was performing. If *init* was asleep, it then scans its database to determine if any other actions should be taken, such as respawning the process.

user signal *Init* catches all signals that it is possible for a process to catch. Most signals have specific meaning to *init*, usually requesting it to change its current state in some way. There is one signal, the 'Q' signal, which is used just to waken *init* and cause it to scan its database. This is often issued after a change has been made to the database so that *init* will put the new change into effect immediately. If this was not done, the change would not become effective until *init* had wakened for some other reason. Other than during the initialization phase, it is solely with signals that the system administrator controls the internal level at which *init* is running.

2.4 Normal Operational Behavior

Init scans **/etc/inittab** once or twice for each event which wakes it up. If it is in the *boot* or *powerfail* state, it scans the table once, looking for entries of these types, and then switches itself back to a *normal* state and scans again.

Its first action in the normal state is to scan **/etc/inittab** and remove all processes which are currently active and should not be at the current level. *Init* employs one of two methods when killing its child processes depending on whether it is changing levels or not. If *init* is not changing levels, it forks a child process for each child that needs to be killed, and has that child process send the signals to the process targeted for extinction. Killing a process involves sending it two signals. First a SIGTERM signal (15), is sent so that it can clean up after itself and die gracefully. After waiting the amount of time defined as TWARN (the default value is 20 seconds), a SIGKILL signal (9), is sent, which guarantees that the child will die, if it hasn't done so already. Forking a child to do the killing has the advantage that the main *init* process need not wait for all the processes it is killing to die before beginning the spawning of new processes. The disadvantage is that if many processes were being killed this way, there would be a very real chance of the operating system process table filling up, which causes the *fork* system call to fail. This in turn would upset *init* at the very least and cause it to have to wait anyway. For this reason, when *init* is changing levels, it assumes that it may have many processes to terminate and so it sends the signals itself, waits for the required 20 seconds, and sends the final termination signals, before continuing. Once the old processes have been removed, *init* makes an entry in its accounting files if it is changing levels. At this point it either enters the single-user level or rescans its database looking for processes that need to be spawned at the current level and in the current state. In the normal state of operation *init* is looking for entries whose types are *off*, *once*, *wait*, or *respawn*.

With the completion of the scan of the database in the normal state, *init* is ready to wait for another event. To ensure that a user who just logged off has had his or her files updated to the disk and to insure that the bookkeepping is also updated to the disk, *init* performs a *sync* system call and then pauses until it is awakened again for some new reason.

If *init* finds that it is being requested to switch to the single-user level when it wakens from the pause, it saves all the *ioctl* information about the system console in the file */etc/ioctl.syscon* before proceeding to remove all its other children. It does this so that if the system is being taken down, the new *init* process will know how to set up the system console to talk to it. It is a convenient feature to not have to change the baud rate and terminal specifications if you are rebooting a system remotely. Because *init* preserves the ioctl state of the system console across system reboots, messages coming out during reboots are legible to the operator, no matter where the system console happens to be linked.

All written messages from *init* are sent to */dev/syscon*. In reality, *init* itself does not send the message, but forks a child to send the message. This is because *init* must never open a terminal line or it will be assigned a controlling terminal. Since *init* has no controlling terminal, it can spawn *getty* processes which initially have no controlling terminal. When such a *getty* opens its assigned terminal, the terminal becomes the controlling terminal for it and its children. In the one instance *init* needs input from the system administrator during the initialization phase. In this case, the child process which is asking for the run level opens */dev/systty*, which is always the physical system console, before opening */dev/syscon*, the virtual system console. This causes */dev/systty* to be the child's controlling terminal. Thus, should the computer be coming up, */dev/syscon* not be linked to */dev/systty*, and */dev/syscon* be down (perhaps because the datalink went down during the reboot), it is possible for a person at */dev/systty* to regain control by typing a <DEL> character. This causes a SIGINT signal (2) to be sent to the child process, which will relink */dev/systty* to */dev/syscon* and ask again for a run level, this time at the physical system console.

2.5 Setting Tunable Variables

Init has several tunable timing constants that can be adjusted when it is compiled.

SLEEPTIME — *Init* guarantees that it will awaken occasionally even if the system is quite inactive. It does this by setting an alarm timer before going to sleep. The length of that timer is defined by SLEEPTIME, and is initially five minutes. Since *init* does a *sync* system call each time it wakes, this guarantees that there will be a *sync* at least once every SLEEPTIME seconds.

TWARN — TWARN is the number of seconds between the SIGTERM signal and the SIGKILL signal, when *init* is removing processes. It should be set long enough so that all processes who want to, can die gracefully on receipt of the SIGTERM signal. It is initially 20 seconds.

NPROC — This is the size of the internal process table *init* uses to keep track of its child processes. It currently defaults to 100, though it can be passed in during compilation with the -D option. I recommend you set it to the size of the system's process table.

WARNFREQUENCY — To prevent *init* from flooding the system console with error messages when it own internal process table is full, *init* only generates an error message once each WARNFREQUENCY times that it is unable to find a slot. Proper sizing of the internal process table should prevent this condition from ever occurring.

Init cannot directly tell if there is something wrong when it tries to fork and exec a command. It assumes that there is something wrong if it has to respawn a particular entry too often. There are three related defines controlling this feature, SPAWN_LIMIT, SPAWN_INTERVAL, and INHIBIT.

SPAWN_LIMIT — SPAWN_LIMIT is the number of times a process may respawn in a certain interval of time before further respawns are inhibited.

SPAWN_INTERVAL — SPAWN_INTERVAL is the interval of time in seconds that SPAWN_LIMIT number of respawns must occur to cause inhibition of an entry. If an entry should respawn too often, a message is generated on the system console indicating which line in **/etc/inittab** is at fault.

INHIBIT — INHIBIT is the number of seconds of inhibition that will be applied to a process which has respawned too often.

SPAWN_LIMIT and SPAWN_INTERVAL should be set so that it is possible for *init* to respawn a process fast enough to cause inhibition, but not so low that it is possible to have a legal death of a process happen so rapidly that it is inhibited. The current limits are ten respawns in two minutes. The real problem is that when something like *getty* disappears, *init* becomes active trying to respawn many processes and never gets to respawn a single process often enough to set off the alarm. The INHIBIT limit is five minutes. Once an entry is inhibited, it is possible to restart it sooner than

INHIBIT seconds later by sending *init* the 'Q' signal. The normal problem is a typo in **/etc/inittab**, and the normal procedure is to correct the typo and then do a "telinit Q" to cause *init* to attempt the spawning entry again.

2.6 Debugging Features

Init has some debugging features built in. There are three conditional debug flags, which allow various flavors of debugging to be enabled.

UDEBUG This flag causes *init* to be compiled in a form that can be run as a normal user process instead of as process 1. This allows a person to use *sdb* on it in a normal fashion and to not disturb the rest of the system while debugging or modifications are made and tested. There are differences in this user version of *init*. It assumes that **utmp**, **wtmp**, **inittab**, **ioctl.syscon**, and **debug** are all in the local directory instead of **/etc**. It also writes to **/dev/sysconx** and **/dev/systtyx**, instead of **/dev/syscon** and **/dev/systty**. It does not process all signals in the same fashion that the real *init* does. Signals SIGINT, SIGQUIT, SIGIOT, and SIGTERM, which correspond to the signals to change to levels 2, 3, 4, and ignore are left in their default modes, so that it is possible to terminate the user "init" from a terminal. Signals SIGUSR1 and SIGUSR2, which are normally ignored by the real *init* are set to cause an *abort* for capturing cores of the debug *init*. The UDEBUG flag automatically sets the DEBUG flag, meaning that the first level of debug will be generated by the *init* and written into the file *debug* in the current directory.

DEBUG This flag causes a version of *init* to be produced that can be run as the real *init*, but which generates diagnostic messages about process removal, level changes, and accounting and writes them in the file **/etc/debug**.

DEBUG1 DEBUG1 causes the diagnostic output generated by DEBUG1 to be increased substantially. Specifically it produces messages about each process being spawned from *inittab*.

3. Getty

Getty is responsible for making appropriate setting of terminal characteristics and baud rate so that a user can communicate with the UNIX system. The most important of those features is the choice of a baud rate so that input and output make sense. In the old version of *getty*, there was a hardwired table in *getty* which controlled the search for the correct speed. The starting point in the search is specified by the arguments passed to *getty*. If there was some reason to change the baud rate search, *getty* had to be modified itself, and recompiled. In the new *getty*, the search is controlled by an ascii file, **/etc/gettydefs**, and changing or augmenting the search behavior only requires that the file be edited.

3.1 Usage

Getty is normally started from **/etc/inittab** by *init*. *Getty* takes from one to six arguments:

getty [-h] [-t time] line [speed_label] [term_type] [line_disc]

-h This switch tells *getty* that it should not drop the Data Terminal Ready signal before resetting the line. This switch currently only works in the CB-UNIX system environment. Normally *getty* ensures that DTR goes down so that connections to the Develcon dataswitch will be disconnected everytime. The EIA protocol requires that a dataset see DTR drop and be reasserted before answering another call. It is possible for *getty* to come back on a line before all the processes spun off by the previous user have died and closed their connections to the line. In this case, DTR would not drop if *getty* didn't insure it. This switch is required for programs like *ct*, which initiate a call from the computer to a user (instead of the user calling the computer), putting a *getty* on the resulting connected line. Without the -h switch, the *getty* would immediately disconnect the user again.

−t — This switch specifies that the *getty* should die after the specified number of seconds if nothing is typed. This prevents datasets from being tied up if someone isn't actually logging in after they've gotten connected.

line — *Line* is the name of the terminal line, which *getty* is to open and set up. It is minus **/dev/** since *getty* does a *chdir* to the **/dev** directory and expects to find it in that directory.

speed_label — The *speed_label* is usually something like "1200" or "9600", which appears to directly specify a baud rate, but in reality can be anything since it really is a label of an entry in **/etc/gettydefs** for which *getty* looks. It specifies the entry *getty* will start with when trying to find an appropriate speed to for the terminal. It defaults to "300" if there is none given.

term_type — The *term_type* specifies which terminal discipline is to be used. If this is specified, the virtual terminal protocol becomes immediately effective on the line. Typical types might be "vt100", "hp45", or "tek". Whatever type is specified, it must be a terminal handler that has been compiled into the operating system to be effective. This argument is given for lines that are hardwired to the computer.

line_disc — The *line discipline* is the last thing that can be specified. The most common is "half" or "half_duplex", when there is a half duplex terminal coming into the computer. This causes the appropriate line discipline to be associated with the line.

3.2 The Database: /etc/gettydefs

Whenever *getty* is invoked it references its database to determine certain information about how to set up the line. Each entry in the database has a fixed format.

label# initial flags # final flags # login msg #nextlabel

*Getty*_matches its *speed_label* argument against the "label" field. It stops searching when it finds an entry with a label that matches. The entry specifies how the terminal is supposed to be setup during the initial phase, the phase when *getty* prints out the "login msg" and reads in the user's login name, and the final phase, when *getty* exec's the *login* program to continue to the *login* process. The baud rate is specified as an *ioctl* flag in both the initial and final flags fields.

The flags themselves are strings matching the define variables found in **/usr/include/termio.h**. It should be noted that these flags may be partially or totally overridden if there is a terminal type specified. When a terminal type is enabled, it resets various flags to suitable conditions automatically.

During the initial phase, *getty* always puts the terminal into a non-echoing *raw* mode. This allows it to take each character as it comes in and infer certain things about the terminal. For instance, if it sees upper case alphabetic characters, but no lower case, it then assumes that the terminal is upper case only and sets it up in the final configuration so that the upper to lower case conversions are made. Also if the speed is wrong it will get a <NULL> character (or <ESC><NULL> character if a terminal type is set) if there is a framing or parity error. This means that the speed is wrong and another speed should be tried.

The typical "initial flags" would only include the speed, for example "B1200 CS7 PARENB HUPCL". "CS7 PARENB" sets the line for 7 bits, even parity characters. "HUPCL sets the line to hangup on close. Typical final flags would be "B1200 SANE IXANY TAB3". "SANE" is not a real flag found in the header file, but a collection of *ioctl* flags used for normal terminal behavior. "IXANY" permits the use of any character to restart output. "TAB3" says to expand tabs on output.

The "login msg" field is the message that *getty* will print before waiting for the user to enter his or her login name. It may contain anything desired and *getty* understands normal special character conventions so that "\n" means <lf> as does "\012". On systems that are not using the terminal

handlers and where lines are hardwired, people have been known to make up special entries for different terminal types, for example:

```
vt100-2400# B2400 # B2400 SANE TAB3 7953OGIN. #vt100-1200
        # 33[H 33[2JAMACCS System B
```

where the "login msg" contains the special vt100 characters required to clear the screen. Notice also that the entry can take more than one line. Entries are delimited by a blank line. Lines that begin with a pound sign (#) are ignored so that comments may be added to the file.

The "next_label" field tells *getty* which entry to try next if it gets an indication that the speed is wrong. In the above example it would look for an entry with the name "vt100-1200" if this one wasn't at the proper speed. Normally the entries don't contain terminal specific information, and the various speed choices are linked together in a closed circle of some sort. For example it is common to have 9600 -> 4800 -> 2400 -> 1200 -> 300 -> 9600. In this way, no matter where you enter the circle, sooner or later you should be able to get to the speed that is correct for your terminal.

To enable the system administrator to check the database for readability by *getty*, there is a checking mode in which *getty* can be run.

```
getty -c gettydefs_like_file
```

When *getty* is run in this mode, it scans the entire input file specified and deciphers each entry, printing out the resulting modes that it will set. If it finds a line that it cannot read, it prints an appropriate message, which allows the administrator to correct the entry. By this mechanism it is possible to avoid installing a misformatted gettydefs file and have it tie up the system.

Also as a safety measure, should *getty* be unable to find **/etc/gettydefs**, it does have a one fallback entry built in. Should *gettydefs* disappear for some reason, a user could still log in at 300 baud, since this is the default setting in the built-in entry.

3.3 Operational Behavior

As has been shown earlier, *getty* sets up a line as specified by an entry from **/etc/gettydefs** and from any additional arguments, outputs the "login msg" field, and then tries to read the user's login name from the line. During the input of the login name, getty checks for speed mismatches that the operating system will report as a <NULL> character. If such a mismatch occurs, *getty* tries the next speed specified by the current entry, and repeats the whole sequence. Also while reading in the login name, *getty* makes a guess whether the terminal is upper case only. If it sees some upper case characters, but no lower case characters, it assumes that the terminal is upper case only and sets the *ioctl* state of the line to translate upper case letters to lower case on input, and lower to upper case on output.

An addition has been made to *getty* and *login*, which allows for environmental variables to be set up at the time a user enters his or her login name. This allows users to control the behavior of their *.profile* at the time they specify their login names. *Getty* executes the *login* program by passing all the separate words given it in response to the login message as arguments to *login*. If for example, the user responded with "jls f", then *getty* would execute "login jls f" as its final action. See the *login* section to see how this modifies the commands behavior.

4. login

Unlike *init* and *getty*, *login* did not require a great deal of modification. The only required change was that it should write to **/etc/utmp** and **/etc/wtmp** in the new format. This change was minor. At the time this change was made, a change visible to the user was also made: the ability to add to the environment. This change was added as a convenience. It allows the user to modify the behavior of his or her *.profile* by having environmental variables set which the *.profile* script knows about.

The basic change was that any additional words provided in response to the basic "login:" query are placed in the environment of the *sh* executed by *login* as its last act in the following way. If the word does not contain an '=', a shell variable of the type "Ln=word" is created. "n" is a number starting at 0 and for each new environment variable it is incremented by one. If the word does contain an '=', then the whole string is passed in the environment unchanged. For example, "TERM=2621" would be placed in the environment unchanged and the shell variable $TERM would be defined as "2621".

To preserve security, there are a couple of exceptions. It is not possible to change the shell variables $PATH or $SHELL by this mechanism. That means that a restricted shell will remain restricted and that the user cannot gain access to commands that might allow him to avoid the usual restrictions of *rsh*.

5. who

Who(1) is the program that reads the history files maintained by *init*, *getty*, and *login*. Since the format of these files was changed substantially, it was necessary to change *who*. In the process some additional features where added to *who* so that it would convey more useful information to users. The standard usage for *who* is:

who [-uTlpdbrtas] [[am i] or [utmp_like_file]]

u This returns a listing of useful information for all the users. This information includes login time, activity, pid and comment from inittab file.

T Report the writability state of the terminal for that entry.

l Report all entries that are living *getty* processes.

p Report all entries for living children of *init* excluding *getty* and decendents of *getty*.

d Report all the entries for processes that have died.

b Report the boot time entries that *init* has made. In **/etc/utmp** there is only one such entry.

r Report the run level entries that *init* has made. In **/etc/utmp** there is only one such entry, the current run level entry. The current state, the number of times in that state, and the previous state are also reported.

t Report the change of date entries that have been made by the *date*(1) command when the clock was reset. These are required in the history file, **/etc/wtmp**, if accounting is to be done.

a Report all the entries.

s Report information for all users in short form, this is the default.

If no file is specified, then **/etc/utmp** is assumed. The **who am i** sequence returns the entry for the user typing the command.

There are various output formats for the different kinds of entry. In particular, entries for users and *getty* processes list the amount of time since output to the terminal occurred. This is often of interest since it shows other users whether someone is actually working at a terminal or not. The comment field at the end of the entry from **/etc/inittab** is also included, which can conveniently be set up to be the location of the terminal. Dead entries report the exit status for the process that died. This can be of use, since it shows whether the process terminated abnormally or not.

6. Other Affected Programs

All programs accessing the accounting files were affected by the new utmp structure. In particular, *date*(1) makes two entries indicating the old time and new time, whenever it changes the system clock. Also affected are the commands in **/usr/lib/acct**, which produces reports based on the information in **/etc/wtmp.**

7. utmp format

A major change in going to the new *init* was that it uses a different format in writing out its records in **/etc/utmp** and **/etc/wtmp.** The new format is:

```
/*      <sys/types.h> must be included.                                 */

#define UTMP_FILE     "/etc/utmp"
#define WTMP_FILE     "/etc/wtmp"
#define ut_name       ut_user

struct utmp
 {
        char ut_user[8] ;       /* User login name */
        char ut_id[4] ;         /* /etc/lines id(usually line #) */
        char ut_line[12] ;              /* device name (console, lnxx) */
        short ut_pid ;                  /* process id */
        short ut_type ;         /* type of entry */
        struct exit_status
          {
           short e_termination ; /* Process termination status */
           short e_exit ;       /* Process exit status */
          }
        ut_exit ;               /* The exit status of a process
                                        * marked as DEAD_PROCESS.
                                        */
        time_t ut_time ;        /* time entry was made */
 } ;

/*      Definitions for ut_type                                         */

#define EMPTY         0
#define RUN_LVL       1
#define BOOT_TIME     2
#define OLD_TIME      3
#define NEW_TIME      4
#define INIT_PROCESS  5       /* Process spawned by "init" */
#define LOGIN_PROCESS 6       /* A "getty" process waiting for login */
#define USER_PROCESS7         /* A user process */
#define DEAD_PROCESS  8
#define ACCOUNTING    9

#define UTMAXTYPE   ACCOUNTING  /* Largest legal value of ut_type */

/*      Special strings or formats used in the "ut_line" field when     */
/*      accounting for something other than a process.                  */
/*      No string for the ut_line field can be more than 11 chars +     */
/*      a NULL in length.                                               */

#define RUNLVL_MSG  "run-level %c"
#define BOOT_MSG    "system boot"
#define OTIME_MSG   "old time"
#define NTIME_MSG   "new time"
```

The *ut_type* field completely identifies the type of entry, the *ut_id* field only contains the "id" as found in the "id" field of **/etc/inittab.** The *ut_line* field was expanded and freed so that it can

contain things like *console* or other things that are not of the form /dev/ln*xx*. Finally *ut_exit* contains the exit status of processes that *init* has spawned and that have subsequently died.

APPENDIX C: SYSTEM ACCOUNTING

The UNIX System Accounting provides methods to collect per-process resource utilization data, record connect sessions, monitor disk utilization, and charge fees to specific logins. A set of C language programs and shell procedures is provided to reduce this accounting data into summary files and reports. This section describes the structure, implementation, and management of this accounting system, as well as a discussion of the reports generated and the meaning of the columnar data.

GENERAL

The following list is a synopsis of the actions of the accounting system:

- At process termination, the UNIX system kernel writes one record per process in */usr/adm/pacct* in the form of *acct.h*. (See Attachment 7.1 for a description of data files.)
- The **login** and **init** programs record connect sessions by writing records into */etc/wtmp*. Date changes, reboots, and shutdowns are also recorded in this file.
- The disk utilization program **acctdusg** breaks down disk usage by login.
- Fees for file restores, etc., can be charged to specific logins with the **chargefee** shell procedure.
- Each day the **runacct** shell procedure is executed via **cron** to reduce accounting data and produce summary files and reports. (See Attachment 7.2 for a sample report output.)
- The **monacct** procedure can be executed on a monthly or fiscal period basis. It saves and restarts summary files, generates a report, and cleans up the *sum* directory. These saved summary files could be used to charge users for UNIX system usage.

FILES AND DIRECTORIES

The */usr/lib/acct* directory contains all of the C language programs and shell procedures necessary to run the accounting system. The *adm* login (currently user ID of four) is used by the accounting system and has the directory structure shown in Fig. 7.1.

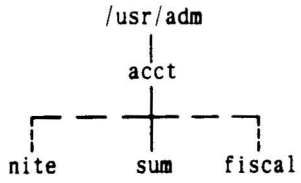


Fig. 7.1—Directory Structure of the "adm" Login

The */usr/adm* directory contains the active data collection files. (For a complete explanation of the files used by the accounting system, see Attachment 7.3.) The *nite* directory contains files that are reused daily by the **runacct** procedure. The *sum* directory contains the cumulative summary files updated by **runacct**. The *fiscal* directory contains periodic summary files created by **monacct**.

DAILY OPERATION

When the UNIX system is switched into multiuser mode, */usr/lib/acct/startup* is executed which does the following:

1. The **acctwtmp** program adds a "boot" record to */usr/adm/wtmp*. This record is signified by using the system name as the login name in the *wtmp* record.

2. Process accounting is started via **turnacct**. **Turnacct on** executes the **accton** program with the argument */usr/adm/pacct*.

3. The **remove** shell procedure is executed to clean up the saved *pacct* and *wtmp* files left in the *sum* directory by **runacct**.

The **ckpacct** procedure is run via **cron** every hour of the day to check the size of */usr/adm/pacct*. If the file grows past 1000 blocks (default), **turnacct switch** is executed. While **ckpacct** is not absolutely necessary, the advantage of having several smaller *pacct* files becomes apparent when trying to restart **runacct** after a failure processing these records.

The **chargefee** program can be used to bill users for file restores, etc. It adds records to */usr/adm/fee* which are picked up and processed by the next execution of **runacct** and merged into the total accounting records.

Runacct is executed via **cron** each night. It processes the active accounting files, */usr/adm/pacct*, */usr/adm/wtmp*, */usr/adm/acct/nite/disktacct*, and */usr/adm/fee*. It produces command summaries and usage summaries by login.

When the system is shut down using **shutdown**, the **shutacct** shell procedure is executed. It writes a shutdown reason record into */usr/adm/wtmp* and turns process accounting off.

After the first reboot each morning, the computer operator should execute */usr/lib/acct/prdaily* to print the previous day's accounting report.

SETTING UP THE ACCOUNTING SYSTEM

In order to automate the operation of this accounting system, several things need to be done:

1. If not already present, add this line to the */etc/rc* file in the state 2 section:

```
/bin/su - adm -c /usr/lib/acct/startup
```

2. If not already present, add this line to */etc/shutdown* to turn off the accounting before the system is brought down:

```
/usr/lib/acct/shutacct
```

3. For most installations, the following three entries should be made in */usr/lib/crontab* so that **cron** will automatically run the daily accounting.

```
"0 4 * * 1-6 /bin/su -adm -c " /usr/lib/acct/runacct
        2> /usr/adm/acct/nite/fd2log "
0 2 * * 4 /usr/lib/acct/dodisk
5 * * * * /bin/su -adm -c "/usr/lib/acct/ckpacct "
```

Note that **dodisk** is invoked with superuser privileges of *root* so that directory searching is not road blocked.

4. To facilitate monthly merging of accounting data, the following entry in *crontab* will allow **monacct** to clean up all daily reports and daily total accounting files and deposit one monthly total report and one monthly total accounting file in the *fiscal* directory.

```
15 5 1 * * /bin/su -adm -c /usr/lib/acct/monacct
```

   The above entry takes advantage of the default action of **monacct** that uses the current month's date as the suffix for the file names. Notice that the entry is executed at such a time as to allow **runacct** sufficient time to complete. This will, on the first day of each month, create monthly accounting files with the entire month's data.

5. The *PATH* shell variable should be set in */usr/adm/.profile* to:

```
PATH=/usr/lib/acct:/bin:/usr/bin
```

RUNACCT

Runacct is the main daily accounting shell procedure. It is normally initiated via **cron** during nonprime time hours. **Runacct** processes connect, fee, disk, and process accounting files. It also prepares daily and cumulative summary files for use by **prdaily** or for billing purposes. The following files produced by **runacct** are of particular interest:

nite/lineuse	Produced by **acctcon**, which reads the *wtmp* file, and produces usage statistics for each terminal line on the system. This report is especially useful for detecting bad lines. If the ratio between the number of logoffs to logins exceeds about 3/1, there is a good possibility that the line is failing.
nite/dayacct	This file is the total accounting file for the previous day in **tacct.h** format.
sum/tacct	This file is the accumulation of each day's *nite/daytacct* which can be used for billing purposes. It is restarted each month or fiscal period by the **monacct** procedure.
sum/daycms	Produced by the **acctcms** program, it contains the daily command summary. The ASCII version of this file is *nite/daycms*.
sum/cms	The accumulation of each day's command summaries. It is restarted by the execution of **monacct**. The ASCII version is *nite/cms*.
sum/loginlog	Produced by the **lastlogin** shell procedure, it maintains a record of the last time each login was used.
sum/rprt.MMDD	Each execution of **runacct** saves a copy of the output of **prdaily**.

Runacct takes care not to damage files in the event of errors. A series of protection mechanisms are used that attempt to recognize an error, provide intelligent diagnostics, and terminate processing in such a way that **runacct** can be restarted with minimal intervention. It records its progress by writing descriptive messages into the file *active*. (Files used by **runacct** are assumed to be in the *nite* directory unless otherwise noted.) All diagnostics output during the execution of **runacct** is written into *fd2log*. To prevent multiple invocations, in the event of two **crons** or other problems, **runacct** will complain if the files *lock* and *lock1* exist when invoked. The *lastdate* file contains the month and day **runacct** was last invoked and is used to prevent more than one execution per day. If **runacct** detects an error, a message is written to */dev/console*, mail is sent to *root* and *adm*, the locks are removed, diagnostic files are saved, and execution is terminated.

In order to allow **runacct** to be restartable, processing is broken down into separate reentrant states. This is accomplished by using a **case** statement inside an endless **while** loop. Each state is one case of the **case**

statement. A file is used to remember the last state completed. When each state completes, *statefile* is updated to reflect the next state. In the next loop through the **while**, *statefile* is read and the **case** falls through to the next state. When **runacct** reaches the **CLEANUP** state, it removes the locks and terminates. States are executed as follows:

SETUP	The command **turnacct switch** is executed. The process accounting files, */usr/adm/pacct?*, are moved to */usr/adm/Spacct?.MMDD*. The */usr/adm/wtmp* file is moved to */usr/adm/acct/nite/wtmp.MMDD* with the current time added on the end.
WTMPFIX	The *wtmp* file in the *nite* directory is checked for correctness by the **wtmpfix** program. Some date changes will cause **acctcon1** to fail, so **wtmpfix** attempts to adjust the time stamps in the *wtmp* file if a date change record appears.
CONNECT1	Connect session records are written to *ctmp* in the form of **ctmp.h**. The *lineuse* file is created, and the *reboots* file is created showing all of the boot records found in the *wtmp* file.
CONNECT2	*Ctmp* is converted to *ctacct.MMDD* which are connect accounting records. (Accounting records are in **tacct.h** format.)
PROCESS	The **acctprc1** and **acctprc2** programs are used to convert the process accounting files, */usr/adm/Spacct?.MMDD*, into total accounting records in *ptacct?.MMDD*. The *Spacct* and *ptacct* files are correlated by number so that if **runacct** fails, the unnecessary reprocessing of *Spacct* files will not occur. One precaution should be noted; when restarting **runacct** in this state, remove the last *ptacct* file because it will not be complete.
MERGE	Merge the process accounting records with the connect accounting records to form *daytacct*.
FEES	Merge in any ASCII *tacct* records from the file *fee* into *daytacct*.
DISK	On the day after the **sdisk** procedure runs, merge *disktacct* with *daytacct*.
MERGETACCT	Merge *daytacct* with *sum/tacct*, the cumulative total accounting file. Each day, *daytacct* is saved in *sum/tacctMMDD*, so that *sum/tacct* can be recreated in the event it becomes corrupted or lost.
CMS	Merge in today's command summary with the cumulative command summary file *sum/cms*. Produce ASCII and internal format command summary files.
USEREXIT	Any installation dependent (local) accounting programs can be included here.
CLEANUP	Clean up temporary files, run **prdaily** and save its output in *sum/rprtMMDD*, remove the locks, then exit.

RECOVERING FROM FAILURE

The **runacct** procedure can fail for a variety of reasons; usually due to a system crash, */usr* running out of space, or a corrupted *wtmp* file. If the *activeMMDD* file exists, check it first for error messages. If the *active* file and lock files exist, check *fd2log* for any mysterious messages. The following are error messages produced by **runacct**, and the recommended recovery actions:

ERROR: locks found, run aborted

The files *lock* and *lock1* were found. These files must be removed before **runacct** can restart.

ERROR: acctg already run for *date*: check /usr/adm/acct/nite/lastdate

The date in *lastdate* and today's date are the same. Remove *lastdate*.

ERROR: turnacct switch returned rc=?

Check the integrity of **turnacct** and **accton**. The **accton** program must be owned by *root* and have the *setuid* bit set.

ERROR: Spacct?.*MMDD* already exists

File setups probably already run. Check status of files, then run setups manually.

ERROR: /usr/adm/acct/nite/wtmp.*MMDD* already exists, run setup manually

Self-explanatory.

ERROR: wtmpfix errors see /usr/adm/acct/nite/wtmperror

Wtmpfix detected a corrupted *wtmp* file. Use **fwtmp** to correct the corrupted file.

ERROR: connect acctg failed: check /usr/adm/acct/nite/log

The **acctcon1** program encountered a bad *wtmp* file. Use **fwtmp** to correct the bad file.

ERROR: Invalid state, check /usr/adm/acct/nite/active

The file *statefile* is probably corrupted. Check *statefile* and read *active* before restarting.

RESTARTING RUNACCT

Runacct called without arguments assumes that this is the first invocation of the day. The argument *MMDD* is necessary if **runacct** is being restarted and specifies the month and day for which **runacct** will rerun the accounting. The entry point for processing is based on the contents of *statefile*. To override *statefile*, include the desired state on the command line. For example:

To start **runacct**:

```
nohup runacct  2>  /usr/adm/acct/nite/fd2log&
```

To restart **runacct**:

```
nohup runacct 0601  2>  /usr/adm/acct/nite/fd2log&
```

To restart **runacct** at a specific state:

```
nohup runacct 0601 WTMPFIX  2>  /usr/adm/acct/nite/fd2log&
```

FIXING CORRUPTED FILES

Unfortunately, this accounting system is not entirely fool proof. Occasionally, a file will become corrupted or lost. Some of the files can simply be ignored or restored from the file save backup. However, certain files must be fixed in order to maintain the integrity of the accounting system.

A. Fixing WTMP Errors

The *wtmp* files seem to cause the most problems in the day to day operation of the accounting system. When the date is changed and the UNIX system is in multiuser mode, a set of date change records is written into */usr/adm/wtmp*. The **wtmpfix** program is designed to adjust the time stamps in the *wtmp* records when a date change is encountered. Some combinations of date changes and reboots, however, will slip through **wtmpfix** and cause **acctcon1** to fail. The following steps show how to patch up a *wtmp* file.

```
cd /usr/adm/acct/nite
fwtmp < wtmp.MMDD > xwtmp
ed xwtmp
   delete corrupted records or
   delete all records from beginning up to the date change
fwtmp -ic < xwtmp > wtmp.MMDD
```

If the *wtmp* file is beyond repair, create a null *wtmp* file. This will prevent any charging of connect time. **Acctprc1** will not be able to determine which login owned a particular process, but it will be charged to the login that is first in the password file for that user id.

B. Fixing TACCT Errors

If the installation is using the accounting system to charge users for system resources, the integrity of *sum/tacct* is quite important. Occasionally, mysterious *tacct* records will appear with negative numbers, duplicate user IDs, or a user ID of 65,535. First check *sum/tacctprev* with **prtacct**. If it looks all right, the latest *sum/tacct.MMDD* should be patched up, then *sum/tacct* recreated. A simple patchup procedure would be:

```
cd /usr/adm/acct/sum
acctmerg -v < tacct.MMDD > xtacct
ed xtacct
   remove the bad records
   write duplicate uid records to another file
acctmerg -i < xtacct > tacct.MMDD
acctmerg tacctprev < tacct.MMDD > tacct
```

Remember that the **monacct** procedure removes all the *tacct.MMDD* files; therefore, *sum/tacct* can be recreated by merging these files together.

UPDATING PNPSPLIT

The **pnpsplit** subroutine is used by **acctcon1** and **acctprc1** to determine the difference between prime and nonprime time. Prime time is defaulted from 9:00 am to 5:00 pm, Monday through Friday. Nonprime time is considered to be all other hours and the entire day for those days listed in the **holidays** structure in *pnpsplit.c*. The holidays listed are accurate for Bell Laboratories New Jersey locations for the year the operating system was released. Every year on the day after Christmas (the last holiday of the calendar year), the following message will be printed on the system console terminal and appear in *log*.

```
*** RECOMPILE pnpsplit WITH NEW HOLIDAYS ***
```

This message will continue to be sent each time the accounting is run until **pnpsplit**, **acctcon1**, and **acctprc1** are recompiled. The following steps should be taken to successfully recompile these programs.

1. Edit *pnpsplit.c* to change the *thisyear* variable to the new year. Update the *holidays* structure to reflect the new holidays. The numeric entry in the structure is the day of the year, less one. For example, New Year's Day (January 1) is entered as 0. *Pnpsplit.c* is in */usr/src/cmd/acct/lib*.

2. Update the accounting library *a.a* and recompile **acctprc1**, and **acctcon1** by:

superuser to root

ARGS= " acctcon1 acctprc1 " /usr/src/:mkcmd acct

DAILY REPORTS

Runacct generates five basic reports upon each invocation. Samples of these reports are shown in Attachment 7.2. They cover the areas of connect accounting, usage by person on a daily basis, command usage reported by daily and monthly totals, and a report of the last time users were logged in.

The following paragraphs describe the reports and the meanings of their tabulated data.

A. Daily Report

In the first part of the report, the **from/to** banner should alert the administrator to the period reported on. The times are the time the last accounting report was generated until the time the current accounting report was generated. It is followed by a log of system reboots, shutdowns, power fail recoveries, and any other record dumped into */usr/adm/wtmp* by the **acctwtmp** program [see **acct**(1M) in the UNIX System Administrator's Manual].

The second part of the report is a breakdown of line utilization. The TOTAL DURATION tells how long the system was in multiuser state (able to be accessed through the terminal lines). The columns are:

LINE	The terminal line or access port.
MINUTES	The total number of minutes that line was in use during the accounting period.
PERCENT	The total number of MINUTES the line was in use divided into the TOTAL DURATION.
# SESS	The number of times this port was accessed for a **login**(1) session.
# ON	This column does not have much meaning anymore. It used to give the number of times that the port was used to log a user on; but since **login**(1) can no longer be executed explicitly to log a new user in, this column should be identical with SESS.
# OFF	This column reflects not just the number of times a user logged off but also any interrupts that occur on that line. Generally, interrupts occur on a port when the **getty**(8) is first invoked when the system is brought to multiuser state. These interrupts occur at a rate of about two per event; therefore, it is not uncommon to see in excess of twice the amount of OFF than ON or SESS. Where this column does come into play is when the # OFF exceeds the # ON by a large factor. This usually indicates that the multiplexer, modem or cable is going bad, or there is a bad connection somewhere. The most common cause of this is an unconnected cable dangling from the multiplexer.

During real time, */usr/adm/wtmp* should be monitored as this is the file that the connect accounting is geared from. If it grows rapidly, execute **acctcon1** to see which tty line is the most noisy. If the interrupting is occurring at a furious rate, general system performance will be effected.

B. Daily Usage Report

This report gives a by-user breakdown of system resource utilization. Its data consists of:

UID	The user ID.

LOGIN NAME — The login name of the user; there can be more than one login name for a single user ID, this identifies which one.

CPU (MINS) — This represents the amount of time the user's process used the central processing unit. This category is broken down into PRIME and NPRIME (nonprime) utilization. The accounting system's idea of this breakdown is located in the accounting library function **pnpsplit** where the **holidays** array, which also determines nonprime time, is also defined. As delivered, prime time is defined to be 0900-1700 hours. The **holidays** array is correct for Bell Laboratories New Jersey locations for the year of the release.

KCORE-MINS — This represents a cummulative measure of the amount of memory a process uses while running. The amount shown reflects kilobyte segments of memory used per minute. This measurement is also broken down into PRIME and NPRIME amounts.

CONNECT (MINS) — This identifies "Real Time" used. What this column really identifies is the amount of time that a user was logged into the system. If this time is rather high and the later column called # OF PROCS is low, this user is what is called a "line hog". That is, this person logs in first thing in the morning and does not hardly touch the terminal the rest of the day. Watch out for these kind of users. This column is also subdivided into PRIME and NPRIME utilization.

DISK BLOCKS — When the disk accounting programs have been run, their output is merged into the total accounting record (*tacct.h*) and shows up in this column. This disk accounting is accomplished by the program **acctdusg**.

OF PROCS — This column reflects the number of processes that was invoked by the user. This is a good column to watch for large numbers indicating that a user may have a shell procedure that runs amock. The most common example of this is for a **crontab** entry to try to execute a user's *.profile* via **su-** that unfortunately prompts for a terminal type and sits in an endless loop trying to read from the terminal (there is not one when **cron** is executing a process). Preventive coding is encouraged in the *.profile*.

OF SESS — This is how many times the user logged onto the system.

DISK SAMPLES — This indicates how many times the disk accounting was run to obtain the average number of DISK BLOCKS listed earlier.

FEE — An often unused field in the total accounting record, the FEE represents the total accumulation of widgets charged against the user by the **chargefee** shell procedure [see **acctsh**(1M)]. The **chargefee** procedure is used to levy charges against a user for special services performed such as file restores, tape manipulation by operators, etc.

C. Daily Command and Monthly Total Command Summaries

These two reports are virtually the same except that the Daily Command Summary only reports on the current accounting period while the Monthly Total Command Summary tells the story for the start of the fiscal period to the current date. In other words, the monthly report reflects the data accumulated since the last invocation of **monacct**.

The data included in these reports gives an administrator an idea as to the heaviest used commands; and based on those commands' characteristics of system resource utilization, a hint as to what to weigh more heavily when system tuning.

These reports are sorted by TOTAL KCOREMIN which is an arbitrary yardstick, but often a good one for calculating "drain" on a system.

COMMAND NAME This is the name of the command. Unfortunately, all shell procedures are lumped together under the name **sh** since only object modules are reported by the process accounting system. The administrator should monitor the frequency of programs called **a.out** or **core** or any other name that does not seem quite right. Often people like to work on their favorite version of backgammon only they do not want everyone to know about it. **Acctcom** is also a good tool to use for determining who executed a suspiciously named command and also if superuser privileges were used.

NUMBER CMDS This is the total number of invocations of this particular command.

TOTAL KCOREMIN The total cummulative measurement of the amount of kilobyte segments of memory used by a process per minute of run time.

TOTAL CPU-MIN The total processing time this program has accumulated.

TOTAL REAL-MIN The total real-time (wall-clock) minutes this program has accumulated. This total is the actual "waited for" time as opposed to kicking off a process in the background.

MEAN SIZE-K This is the mean of the TOTAL KCOREMIN over the number of invocations reflected by NUMBER CMDS.

MEAN CPU-MIN This is the mean derived between the NUMBER CMDS and TOTAL CPU-MIN.

HOG FACTOR This is a relative measurement of the ratio of system availability to system utilization. It is computed by the formula

(total CPU time) / (elapsed time)

This gives a relative measure of the total available CPU time consumed by the process during its execution.

CHARS TRNSFD This column, which may go negative, is a total count of the number of characters pushed around by the **read**(2) and **write**(2) system calls.

BLOCKS READ A total count of the physical block reads and writes that a process performed.

D. Last Login

This report simply gives the date when a particular login was last used. This could be a good source for finding likely candidates for the tape archives or getting rid of unused logins and login directories.

SUMMARY

The UNIX System Accounting was designed from a UNIX system administrator's point of view. Every possible precaution has been taken to ensure that the system will run smoothly and without error. It is important to become familiar with the C programs and shell procedures. The manual pages should be studied, and it is advisable to keep a printed copy of the shell procedures handy. The accounting system should be easy to maintain, provide valuable information for the administrator, and provide accurate breakdowns of the usage of system resources for charging purposes.

Format of wtmp files (utmp.h):

```
/*      %W%     */

/*      <sys/types.h> must be included.                                 */

#define UTMP_FILE       "/etc/utmp"
#define WTMP_FILE       "/etc/wtmp"
#define ut_name ut_user

struct  utmp
  {
        char ut_user[8] ;               /* User login name */
        char ut_id[4] ;                 /* /etc/lines id(usually line #) */
        char ut_line[12] ;              /* device name (console, lnxx) */
        short ut_pid ;                  /* process id */
        short ut_type ;                 /* type of entry */
        struct exit_status
          {
            short e_termination ;       /* Process termination status */
            short e_exit ;              /* Process exit status */
          }
        ut_exit ;                       /* The exit status of a process
                                         * marked as DEAD_PROCESS.
                                         */
        time_t ut_time ;                /* time entry was made */

    } ;
/*      Definitions for ut_type                                         */

#define EMPTY           0
#define RUN_LVL         1
#define BOOT_TIME       2
#define OLD_TIME        3
#define NEW_TIME        4
#define INIT_PROCESS    5       /* Process spawned by "init" */
#define LOGIN_PROCESS   6       /* A "getty" process waiting for login */
#define USER_PROCESS    7       /* A user process */
#define DEAD_PROCESS    8
#define ACCOUNTING      9

#define UTMAXTYPE       ACCOUNTING      /* Largest legal value of ut_type */

/*      Special strings or formats used in the "ut_line" field when     */
/*      accounting for something other than a process.                  */
/*      No string for the ut_line field can be more than 11 chars +     */
/*      a NULL in length.                                               */

#define RUNLVL_MSG      "run-level %c"
#define BOOT_MSG        "system boot"
#define OTIME_MSG       "old time"
#define NTIME_MSG       "new time"
```

Definitions (acctdef.h):

```
/*      %W% of %G%      */
/*
 *      defines, typedefs, etc. used by acct programs
 */


/*
 *      acct only typedefs
 */
typedef unsigned short  uid_t;

#ifdef u3b
#define HZ      100
#else
#define HZ      60
#endif

#define LSZ     12      /* sizeof line name */
#define NSZ     8       /* sizeof login name */
#define P       0       /* prime time */
#define NP      1       /* nonprime time */

/*
 *      limits which may have to be increased if systems get larger
 */
#define SSIZE   1000    /* max number of sessions in 1 acct run */
#define TSIZE   100     /* max number of line names in 1 acct run */
#define USIZE   500     /* max number of distinct login names in 1 acct run */

#define EQN(s1, s2)             (strncmp(s1, s2, sizeof(s1)) == 0)
#define CPYN(s1, s2)            strncpy(s1, s2, sizeof(s1))

#define SECSINDAY               86400L
#define SECS(tics)              ((double) tics)/HZ
#define MINS(secs)              ((double) secs)/60
#define MINT(tics)              ((double) tics)/(60*HZ)

#ifdef pdp11
#define KCORE(clicks)           ((double) clicks/16)
#endif
#ifdef vax
#define KCORE(clicks)           ((double) clicks/2)
#endif
#ifdef u3b
#define KCORE(clicks)           ((double) clicks*2)
#endif
```

Format of pacct files (acct.h):

```
/*
 * Accounting structures
 */

typedef ushort comp_t;          /* "floating point" */
                /* 13-bit fraction, 3-bit exponent */

struct  acct

        char    ac_flag;        /*Accounting flag */
        char    ac_stat;        /*Exit status */
        ushort  ac_uid;         /*Accounting user ID */
        ushort  ac_gid;         /*Accounting group ID */
        dev_t   ac_tty;         /*control typewriter */
        time_t  ac_btime;       /*Beginning time */
        comp_t  ac_utime;       /*acctng user time in clock ticks */
        comp_t  ac_stime;       /*acctng system time in clock ticks */
        comp_t  ac_etime;       /*acctng elapsed time in clock ticks */
        comp_t  ac_mem;         /*memory usage */
        comp_t  ac_io;          /*chars transferred */
        comp_t  ac_rw;          /*blocks read or written */
        char    ac_comm[8];     /*command name */
;

extern  struct  acct    acctbuf;
extern  struct  inode   *acctp; /*inode of accounting file */

#defineAFORK            01      /*has executed fork, but no exec */
#defineASU              02      /*used superuser privileges */
#defineACCTF            0300    /*record type: 00 = acct */
```

Format of tacct files (tacct.h):

```
/*
 *      total acctounting (for acct period), also for day
 */

struct  tacct
        uid_t           ta_uid;         /*userid */
        char            ta_name[8];     /*login name */
        float           ta_cpu[2];      /*cum. cpu time, p/np (mins) */
        float           ta_kcore[2];    /*cum kcore-minutes, p/np */
        float           ta_con[2];      /*cum. connect time, p/np, mins */
        float           ta_du;          /*cum. disk usage */
        long            ta_pc;          /*count of processes */
        unsigned short  ta_sc;          /*count of login sessions */
        unsigned short  ta_dc;          /*count of disk samples */
        unsigned short  ta_fee;         /*fee for special services */
```

Format of ctmp file (ctmp.h):

```
/*
 *      connect time record (various intermediate files)
 */
struct ctmp
        dev_t   ct_tty;         /*major minor */
        uid_t   ct_uid;         /*userid */
        char    ct_name[8];     /*login name */
        long    ct_con[2];      /*connect time (p/np) secs */
        time_t  ct_start;       /*session start time */
```

Jun 8 04:14 1979 DAILY REPORT FOR pwba Page 1

from Thu Jun 7 06:00:48 1979

to Fri Jun 8 04:00:28 1979

2 shutdown

2 pwa

TOTAL DRATION IS 1320 MINUTES

LINE	MINUTES	PERCENT	# SESS	# ON	# OFF
tty04	479	36	9	9	30
tty47	341	26	4	4	33
tty44	298	23	3	3	29
tty46	336	25	9	9	33
console	1100	83	14	14	21
tty05	448	34	3	3	22
tty06	439	33	9	9	31
tty07	421	32	6	6	24
tty42	53	4	5	5	20
tty09	385	29	11	11	33
tty10	336	25	10	10	31
tty08	464	35	2	2	19
tty26	544	41	6	6	24
tty12	252	19	5	5	25
tty13	258	20	3	3	21
tty14	156	12	6	6	26
tty17	145	11	1	1	16
tty18	39	3	5	5	24
tty15	228	17	5	5	25
tty25	704	53	6	6	25
tty21	0	0	0	0	16
tty19	10	1	1	1	17
tty20	25	2	2	2	18
tty22	0	0	0	0	15
tty23	0	0	0	0	15
tty24	0	0	0	0	16
tty27	481	36	3	3	20
tty28	426	32	5	5	24
tty29	302	23	6	6	25
tty30	257	20	11	11	28
tty40	380	29	5	5	21
tty41	343	26	3	3	21
tty45	0	0	0	0	15
tty11	365	28	7	7	25
tty43	3	0	1	1	17
tty16	213	16	3	3	20
tty31	250	19	4	4	18
tty02	62	5	1	1	3
TOTALS	10544	--	174	174	846

Jun 8 04:14 1979 DAILY USAGE REPORT FOR pwba Page 1

UID	LOGIN NAME	CPU (MINS) PRIME	CPU (MINS) NPRIME	KCORE-MINS PRIME	KCORE-MINS NPRIME	CONNECT (MINS) PRIME	CONNECT (MINS) NPRIME	DISK BLOCKS	# OF PROCS	# OF SESS	# DISK SAMPLES	FEE
0	TOTAL	388	103	12414	2934	9251	1056	0	16164	174	0	0
0	root	47	41	1003	924	67	30	0	2360	8	0	0
4	adm	27	19	48	652	0	0	0	842	0	0	0
19	games	0	0	4	0	0	0	0	28	0	0	0
22	mhb	0	0	1	1	1	1	0	14	2	0	0
37	abs	0	0	4	0	0	0	0	3	0	0	0
37	absjrk	14	0	284	0	423	0	0	1588	4	0	0
68	rje	3	3	24	21	0	0	0	179	0	0	0
71	?	0	0	0	0	0	0	0	12	0	0	0
150	jac	7	0	156	5	281	2	0	510	13	0	0
173	?	0	0	0	0	0	0	0	16	0	0	0
180	?	0	0	0	0	0	0	0	4	0	0	0
185	?	0	0	0	0	0	0	0	2	0	0	0
217	denise	0	0	2	0	31	0	0	32	3	0	0
217	kof	0	0	2	0	1	0	0	7	1	0	0
219	?	0	0	0	0	0	0	0	12	0	0	0
1001	hsm	5	0	189	0	179	0	0	92	2	0	0
2001	systst	0	1	5	28	476	64	0	99	5	0	0
2002	mfp	1	0	7	5	270	62	0	93	3	0	0
2003	als	1	0	23	0	100	0	0	99	3	0	0
2005	eric	0	0	3	0	13	0	0	21	1	0	0
2006	hoot	0	0	2	0	16	0	0	8	1	0	0
2009	agp	47	0	2040	0	444	0	0	492	2	0	0
2009	fsrepl	2	0	60	0	36	0	0	95	1	0	0
2011	pdw	0	0	1	0	4	0	0	11	1	0	0
2012	pwbst	0	0	1	0	28	0	0	9	1	0	0
2014	cath	0	0	1	0	1	0	0	7	1	0	0
2022	rem	32	1	1227	91	576	4	0	226	3	0	0
2025	fld	55	23	2176	862	336	98	0	750	7	0	0
2027	krb	14	2	365	51	547	24	0	372	8	0	0
2028	text	0	0	1	0	3	0	0	13	1	0	0
2030	arf	8	0	288	0	317	0	0	315	3	0	0
2031	dp	12	0	480	3	459	6	0	220	6	0	0
2032	graf	2	0	49	0	23	0	0	118	1	0	0
2033	ecp	3	0	74	0	355	0	0	115	4	0	0
2040	leap	15	0	308	0	513	1	0	505	2	0	0
2041	dan	3	0	93	3	149	2	0	117	8	0	0
2051	ds52	2	2	19	40	375	601	0	611	8	0	0
2055	nuucp	0	0	15	9	17	1	0	10	3	0	0
2057	ech	1	0	28	0	63	0	0	68	2	0	0
2061	jcw	4	3	99	70	37	34	0	869	4	0	0
2064	mjr	18	0	443	0	176	0	0	2065	3	0	0
2065	rrr	0	0	6	0	7	0	0	23	1	0	0
2068	trc	0	0	7	0	10	0	0	29	1	0	0
2075	herb	29	0	1178	1	384	2	0	249	5	0	0
2086	paul	1	0	14	0	152	0	0	28	1	0	0
2087	pris	0	0	0	10	0	2	0	13	1	0	0
2111	pwvcs	2	3	60	85	64	86	0	185	1	0	0
2116	rbj	1	0	16	0	408	0	0	222	1	0	0
2121	teach	0	0	3	0	53	0	0	50	2	0	0
2123	msb	0	0	3	0	5	0	0	24	1	0	0
2124	rnt	2	0	42	0	66	0	0	260	3	0	0
2126	dal	0	0	5	0	121	0	0	17	1	0	0
2127	m2	15	0	495	11	390	2	0	602	10	0	0

Jun 8 04:14 1979 DAILY USAGE REPORT FOR pwba Page 2

2128	jel	14	0	492	9	422	14	0	523	8	0	0
2130	sl	0	0	5	1	16	0	0	42	2	0	0
2130	s3	0	0	0	0	0	2	0	9	1	0	0
2135	jfn	0	1	0	12	0	11	0	33	2	0	0
2136	m2class	0	0	5	0	2	0	0	18	1	0	0
2140	star	4	0	213	12	90	3	0	170	7	0	0
2141	reg	5	0	245	25	470	4	0	181	1	0	0
2199	llc	0	0	1	0	10	0	0	7	1	0	0
2999	stock	0	0	1	0	1	0	0	17	1	0	0
3001	whm	5	0	93	0	253	0	0	414	3	0	0
3332	vjf	0	0	4	0	8	0	0	39	1	0	0

Jun 8 04:07 1979 DAILY COMMAND SUMMARY Page 1

COMMAND NAME	NUMBER CMDS	TOTAL KCOREMIN	TOTAL CPU-MIN	TOTAL REAL-MIN	MEAN SIZE-K	MEAN CPU-MIN	HOG FACTOR	CHARS TRNSFD	BLOCKS READ
TOTALS	16164	15332.89	490.72	37463.98	31.25	003	0.01	322183844	1097670
nroff	119	3958.68	93.21	569.83	42.47	0.78	0.16	67070052	130284
troff	26	2483.38	51.63	342.70	48.10	1.99	0.15	37869304	48989
xnroff	20	732.03	16.74	111.05	43.73	0.84	0.15	13885248	22659
a.out	31	623.53	10.52	142.77	59.26	0.34	0.07	382435	2758
egrep	185	574.83	13.96	34.53	41.18	0.08	0.40	170625	8249
m2fins	232	555.79	9.93	155.11	55.96	0.04	0.06	6155937	30994
c1	150	519.04	13.57	48.89	38.25	0.09	0.28	4285724	16032
c0	165	413.10	9.19	35.16	44.93	0.06	0.26	3827309	12170
m2edit	33	340.92	4.63	148.27	73.62	0.14	0.03	1074914	14492
ld	87	317.38	7.94	38.48	39.97	0.09	0.21	17640896	45797
acctcms	17	294.75	6.49	14.15	45.41	0.38	0.46	2525427	5515
c2	112	289.69	9.13	34.61	31.72	0.08	0.26	3667050	9681
sh	1834	276.98	26.77	20444.24	10.35	0.01	0.00	3496613	71979
ed	524	253.13	14.46	2029.89	17.50	0.03	0.01	13058108	56039
acctprc1	3	231.28	6.67	19.45	34.67	2.22	0.34	2577344	2926
du	145	219.35	19.91	39.08	11.02	0.14	0.51	716389	23695
diff	49	175.53	6.04	25.78	29.05	0.12	0.23	3740887	11351
get	151	152.96	4.28	25.23	35.74	0.03	0.17	3634042	24917
adb	22	148.10	4.07	202.35	36.37	0.19	0.02	2313718	9813
tbl	24	143.43	2.44	210.65	58.71	0.10	0.01	1536210	3433
dd	9	139.24	10.15	51.05	13.72	1.13	0.20	26006848	294
as2	155	129.33	9.82	42.25	13.17	0.06	0.23	10500835	30165
sed	597	115.46	4.19	36.23	27.57	0.01	0.12	783825	24497
ps	51	109.69	5.92	41.55	18.54	0.12	0.14	2278056	8310
make	89	102.94	2.87	203.32	35.81	0.03	0.01	1018461	8664
delta	25	90.23	2.27	17.80	39.70	0.09	0.13	2909269	9321
cpp	172	89.37	2.69	11.32	33.19	0.02	0.24	3519054	12155
fsck	16	86.94	1.30	10.57	66.85	0.08	0.12	27671849	2927
find	52	86.64	5.05	63.87	17.15	0.10	0.08	565125	11161
ls	706	82.47	5.78	62.85	14.26	0.01	0.09	1811882	29659
xck	2	79.44	10.49	47.89	7.57	5.25	0.22	198016	21995
awk	22	78.83	1.37	5.24	57.72	0.06	0.26	355466	3769
uucico	60	75.55	1.42	632.50	53.27	0.02	0.00	398693	6377
acctcom	9	75.21	2.85	11.49	26.75	0.31	0.24	1283776	3771
echo	2814	66.10	7.08	91.80	9.33	0.00	0.08	168651	24253
ged	3	57.27	0.82	7.51	70.16	0.27	0.11	51832	426
dc	284	56.92	2.42	9.43	23.48	0.01	0.26	14283	20329
450	7	48.03	6.80	84.45	7.06	0.97	0.08	279451	1700
cat	749	45.49	5.69	478.54	8.00	0.01	0.01	3959500	27903
ntd	6	41.52	1.55	7.55	26.87	0.26	0.20	59888	478
mail	202	39.95	2.05	532.98	19.53	0.01	0.00	427217	14377
acctprc2	3	38.95	1.43	19.45	27.24	0.48	0.07	587336	87
sort	94	38.72	1.09	9.73	35.41	0.01	0.11	375876	4433
pr	104	34.89	2.47	214.50	14.10	0.02	0.01	1060989	6572
haspmain	7	33.20	5.28	1244.54	6.29	0.75	0.00	63064	36635
ex	17	31.69	0.62	41.04	50.97	0.04	0.02	514624	3593
grep	213	28.73	2.98	21.01	9.64	0.01	0.14	2100229	14297

Jun 8 04:07 1979 MONTHLY TOTAL SUMMARY Page 1

COMMAND NAME	NUMBER CMDS	TOTAL KCOREMIN	TOTAL CPU-MIN	TOTAL REAL-MIN	MEAN SIZE-K	MEAN CPU-MIN	HOG FACTOR	CHARS TRNSFD	BLOCKS READ
TOTALS	553286	297698.78	10916.09	742924.94	27.27	0.02	0.01	820472546	26253312
nroff	1687	44681.55	995.92	5737.25	44.86	0.59	0.17	613403153	1089180
troff	1351	25692.15	583.69	4356.05	44.02	0.43	0.13	413163589	646243
spellpro	6466	17298.41	294.16	1893.79	58.81	0.05	0.16	334572640	853901
m2edit	654	13526.69	164.62	4238.58	82.17	0.25	0.04	54940426	427924
xnroff	397	10408.44	203.72	1496.32	51.09	0.51	0.14	215221419	301967
sort	7983	9292.34	226.01	2298.05	41.11	0.03	0.10	80108304	355963
cl	6139	8949.86	236.45	861.09	37.85	0.04	0.27	79897995	489661
ld	3244	8852.96	223.19	1128.09	39.67	0.07	0.20	493701995	1278119
sed	53134	8126.71	313.85	2241.78	25.89	0.01	0.14	23035033	1692990
m2find	2982	7984.45	140.18	1698.25	56.96	0.05	0.08	111330040	449604
c0	6586	7866.42	185.16	725.47	42.49	0.03	0.26	72595655	389426
ed	20083	7822.78	425.90	41898.18	18.37	0.02	0.01	483425634	1541326
tbl	660	7766.69	113.95	2458.55	68.16	0.17	0.05	50760094	83887
sh	40476	7499.67	635.00	383786.53	11.81	0.02	0.00	70525236	1421194
du	1941	6730.54	553.04	1128.44	12.17	0.28	0.49	20848359	628324
a.out	1483	5658.46	126.87	1868.87	44.60	0.09	0.07	16158675	80260
egrep	4801	5573.51	139.86	460.25	39.85	0.03	0.30	6823696	237298
lint1	793	5325.66	71.23	425.67	74.76	0.09	0.17	9599001	131592
cat	21170	4657.53	236.59	4354.24	19.69	0.01	0.05	239180412	1023965
acctprc1	42	3837.84	110.88	291.34	34.61	2.64	0.38	43954136	61123
c2	4067	3807.25	144.86	477.28	26.28	0.04	0.30	57519376	213521
grep	21212	3204.86	300.44	2727.87	10.67	0.01	0.11	139340583	899415
cpp	7469	3060.72	94.12	647.79	32.52	0.01	0.15	91471956	459882
getty	35556	2948.71	853.53	101107.45	3.45	0.02	0.01	34704751	263866
m2editD	83	2707.27	28.79	361.84	94.02	0.35	0.08	2852202	33949
as2	6454	2698.74	218.96	910.59	12.33	0.03	0.24	213336016	705690
make	1858	2449.10	64.69	4388.86	37.86	0.03	0.01	24116259	175544
ps	1034	2384.14	128.29	1207.87	18.58	0.12	0.11	54873792	204172
acctcms	294	2288.36	51.99	116.06	44.01	0.18	0.45	36124940	80523
uucico	815	2226.75	40.42	11729.01	55.08	0.05	0.00	11086105	162558
ls	18876	2170.01	152.76	1538.09	14.20	0.01	0.10	32418106	691028
find	1705	2114.18	114.35	920.75	18.49	0.07	0.12	94631199	338600
ged	72	2026.43	28.54	317.21	71.01	0.40	0.09	1648636	10374
echo	84710	2018.23	190.14	1138.49	10.61	0.00	0.17	2926992	649200
cpio	127	1956.60	77.03	391.45	25.40	0.61	0.20	190822346	296302
maze	8	1620.42	44.80	128.25	36.17	5.60	0.35	120399	212
mail	4735	1474.38	76.92	14262.62	19.17	0.02	0.01	25719618	463748
get	1085	1358.03	37.59	234.97	36.13	0.03	0.16	31540008	178623
acctcom	165	1253.99	47.06	339.34	26.64	0.29	0.14	57405662	68949
yacc	58	1187.17	15.36	36.90	77.31	0.26	0.42	4096070	12093
col	638	1064.40	49.01	2199.00	21.72	0.08	0.02	23835395	16903
line	27184	1036.03	93.14	1941.33	11.12	0.00	0.05	925447	296142
nroff1.2	29	909.83	17.71	56.97	51.38	0.61	0.31	11459920	18802
delta	264	904.54	23.07	254.06	39.21	0.09	0.09	24219141	87164
td	175	886.19	25.74	159.73	34.43	0.15	0.16	1990177	15792
ar	1434	872.65	61.87	309.07	14.11	0.04	0.20	189858731	428871
m2findD	144	864.29	12.54	344.13	68.94	0.09	0.04	1184947	28576
rm	15319	857.97	85.65	754.20	10.02	0.01	0.11	453479	433903
acctdusg	1	819.77	39.30	170.10	20.86	39.30	0.23	1812480	39744
f77pass1	155	779.13	7.97	29.09	97.70	0.05	0.27	990027	34702
diff	786	767.31	32.77	260.27	23.41	0.04	0.13	22940094	97214

Jun 8 04:07 1979 LAST LOGIN Page 1

00-00-00	dii	00-00-00	rudd	79-06-08	adm
00-00-00	absadm	00-00-00	s10	79-06-08	agp
00-00-00	absafr	00-00-00	s2	79-06-08	als
00-00-00	abscas	00-00-00	s4	79-06-08	arf
00-00-00	absjcw	00-00-00	s5	79-06-08	cath
00-00-00	abspvg	00-00-00	s6	79-06-08	dal
00-00-00	abstbm	00-00-00	s8	79-06-08	dan
00-00-00	adm94	00-00-00	s9	79-06-08	denise
00-00-00	apb	00-00-00	scbsa	79-06-08	dp
00-00-00	archive	00-00-00	sjm	79-06-08	ds52
00-00-00	asc	00-00-00	srb	79-06-08	ech
00-00-00	badt	00-00-00	sys	79-06-08	ecp
00-00-00	btb	00-00-00	tgp	79-06-08	eric
00-00-00	bvl	00-00-00	tld	79-06-08	fld
00-00-00	bwk	00-00-00	ussc	79-06-08	fsrepl
00-00-00	chicken	00-00-00	uucpa	79-06-08	games
00-00-00	class	00-00-00	uvac	79-06-08	graf
00-00-00	cleary	00-00-00	vav	79-06-08	herb
00-00-00	cs	00-00-00	wdr	79-06-08	hoot
00-00-00	dbs	00-00-00	willa	79-06-08	hsm
00-00-00	deby	00-00-00	zooma	79-06-08	jac
00-00-00	dec	79-06-04	dws	79-06-08	jcw
00-00-00	demo	79-06-04	ewb	79-06-08	jel
00-00-00	dlt	79-06-04	kas	79-06-08	jfn
00-00-00	dmr	79-06-04	satz	79-06-08	kof
00-00-00	docs	79-06-04	uucp	79-06-08	krb
00-00-00	dug	79-06-05	bcm	79-06-08	leap
00-00-00	ellie	79-06-05	lprem	79-06-08	llc
00-00-00	fsrep2	79-06-05	s7	79-06-08	m2
00-00-00	gas	79-06-05	sccs	79-06-08	m2class
00-00-00	graphics	79-06-06	conv	79-06-08	mfp
00-00-00	hjg	79-06-06	dck	79-06-08	mhb
00-00-00	hlb	79-06-06	dmt	79-06-08	mjr
00-00-00	inst	79-06-06	emp	79-06-08	msb
00-00-00	jfm	79-06-06	pah	79-06-08	nuucp
00-00-00	jrh	79-06-06	sync	79-06-08	paul
00-00-00	ken	79-06-06	tad	79-06-08	pdw
00-00-00	lco	79-06-07	ams	79-06-08	pris
00-00-00	learn	79-06-07	bin	79-06-08	pwbcs
00-00-00	lppdw	79-06-07	dgd	79-06-08	pwbst
00-00-00	lrbb	79-06-07	haight	79-06-08	rbj
00-00-00	maj	79-06-07	hasp	79-06-08	reg
00-00-00	mar	79-06-07	jgw	79-06-08	rem
00-00-00	mash	79-06-07	leb	79-06-08	rje
00-00-00	meq	79-06-07	ljk	79-06-08	rnt
00-00-00	mifi	79-06-07	mep	79-06-08	root
00-00-00	mlc	79-06-07	nhg	79-06-08	rrr
00-00-00	mmr	79-06-07	nws	79-06-08	s1
00-00-00	mpf	79-06-07	qtroff	79-06-08	s3
00-00-00	plan	79-06-07	tbm	79-06-08	star
00-00-00	plum	79-06-07	train	79-06-08	stock
00-00-00	pvg	79-06-07	whr	79-06-08	systst
00-00-00	rakesh	79-06-07	wwe	79-06-08	teach
00-00-00	rfg	79-06-03	?	79-06-08	text
00-00-00	rlc	79-06-08	abs	79-06-08	trc
00-00-00	rrc	79-06-08	absjrk	79-06-08	vjf
				79-06-08	whm

Files in the /usr/adm directory:

diskdiag	diagnostic output during the execution of disk accounting programs
dtmp	output from the **acctdusg** program
fee	output from the **chargefee** program, ASCII tacct records
pacct	active process accounting file
pacct?	process accounting files switched via ***turnacct***
Spacct?.MMDD	process accounting files for **MMDD** during execution of **runacct**
wtmp	active wtmp file for recording connect sessions

Files in the /usr/adm/acct/nite directory:

active	used by **runacct** to record progress and print warning and error messages; *active* **MMDD** same as *active* after **runacct** detects an error
cms	ASCII total command summary used by **prdaily**
ctacct.MMDD	connect accounting records in **tacct.h** format
ctmp	output of **acctconl** program, connect session records in **ctmp.h** format
daycms	ASCII daily command summary used by **prdaily**
dayacct	total accounting records for one day in **tacct.h** format
disktacct	disk accounting records in **tacct.h** format, created by **dodisk** procedure
fd2log	diagnostic output during execution of runacct (see **cron** entry)
lastdate	last day **runacct** executed in **date +%m%d** format
lock lock1	used to control serial use of **runacct**
lineuse	tty line usage report used by **prdaily**
log	diagnostic output from **acctconl**
logMMDD	same as *log* after **runacct** detects an error
reboots	contains beginning and ending dates from *wtmp*, and a listing of reboots
statefile	used to record current state during execution of **runacct**
tmpwtmp	wtmp file corrected by **wtmpfix**
wtmperror	place for **wtmpfix** error messages
wtmperrorMMDD	same as *wtmperror* after **runacct** detects an error
wtmp.MMDD	previous day's wtmp file

Files in the /usr/adm/acct/sum directory:

cms	total command summary file for current fiscal in internal summary format
cmsprev	command summary file without latest update
daycms	command summary file for yesterday in internal summary format
loginlog	created by **lastlogin**
pacct.MMDD	concatenated version of all pacct files for **MMDD**, removed after reboot by **remove** procedure
rprt.MMDD	saved output of **prdaily** program
tacct	cumulative total accounting file for current fiscal
tacctprev	same as *tacct* without latest update
tacct.MMDD	total accounting file for **MMDD**
wtmp.MMDD	saved copy of wtmp file for **MMDD**, removed after reboot by **remove** procedure

Files in the /usr/adm/acct/fiscal directory:

cms?	total command summary file for fiscal ? in internal summary format
fiscrpt?	report similar to **prdaily** for fiscal ?
tacct?	total accounting file for fiscal ?

APPENDIX D: LP SPOOLING SYSTEM

The LP system of commands performs diverse spooling functions under the operating system. LP allows administrators to customize the system to spool to a collection of printers of any type and to group printers into logical classes in order to maximize the throughput of the devices. Users can queue and cancel print requests, prevent and allow queueing to and print on specific devices, start and stop LP processing requests, change configuration of printers, and find the status of the LP system. This section describes how the Administrator performs restricted functions and oversees LP operation.

OVERVIEW OF LP FEATURES

A. Definitions

Several terms must be defined before presenting a brief summary of LP commands. The LP was designed with the flexibility to meet the needs of users on different UNIX systems. Changes to the LP configuration are performed by the **lpadmin**(1M) command.

LP makes a distinction between printers and printing devices. A *device* is a physical peripheral device or a file and is represented by a full UNIX system pathname. A *printer* is a logical name that represents a device. At different points in time, a printer may be associated with different devices. A *class* is a name given to an ordered list of printers. Every class must contain at least one printer. Each printer may be a member of zero or more classes. A *destination* is a printer or a class. One destination may be designated as the *system default destination.* The **lp**(1) command will direct all output to this destination unless the user specifies otherwise. Output that is routed to a printer will be printed only by that printer, whereas output directed to a class will be printed by the first available class member.

Each invocation of **lp** creates an output request that consists of the files to be printed and options from the lp command line. An interface program which formats requests must be supplied for each printer. The LP scheduler, **lpsched**(1M), services requests for all destinations by routing requests to interface programs to do the printing on devices. An LP configuration for a system consists of devices, destinations, and interface programs.

B. Commands

Commands for General Use

The **lp**(1) command is used to request the printing of files. It creates an output request and returns a request id of the form

dest-seqno

to the user, where *seqno* is a unique sequence number across the entire LP system, and *dest* is the destination where the request was routed.

Cancel is used to cancel output requests. The user supplies request ids as returned by **lp** or printer names, in which case the currently printing requests on those printers are canceled.

Disable prevents **lpsched** from routing output requests to printers.

Enable(1) allows **lpsched** to route output requests to printers.

Commands for LP Administrators

Each LP system must designate a person or persons as LP administrator to perform the restricted functions listed below. Either the superuser or any user who is logged into the UNIX system as **lp** qualifies as an LP Administrator. All LP files and commands are owned by **lp**, except for **lpadmin** and **lpsched** which are owned by root. The following commands will be described in more detail later in this section.

Lpadmin(1M)	Modifies LP configuration. Many features of this command cannot be used when **lpsched** is running.
Lpsched(1M)	Routes output requests to interface programs which do the printing on devices.
Lpshut	Stops **lpsched** from running. All printing activity is halted, but other LP commands may still be used.
Accept(1M)	Allows **lp** to accept output requests for destinations.
Reject	Prevents **lp** from accepting requests for destinations.
Lpmove	Moves output requests from one destination to another. Whole destinations may be moved at once. This command cannot be used when **lpsched** is running.

BUILDING LP

All LP commands are built from source code that resides in the */usr/src/cmd/lp* directory including th make file, *lp.mk*. Unless some of the definitions in *lp.mk* are changed, LP may be installed only by the superuser. Before installing a new LP system, make sure there is a login called *lp* on your system and that the spool directory, */usr/spool/lp*, does not exist. To install LP, perform the following:

```
cd /usr/src/cmd/lp
make -f lp.mk install
```

This builds all LP commands and creates an initial LP configuration consisting of no printers, classes, or default destination. LP must be configured by an LP administrator using the **lpadmin** command in order to create a useful spooler.

In addition, add the following code to */etc/rc*:

```
rm -f /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched
echo " LP scheduler started "
```

This starts the LP scheduler each time that the UNIX system is restarted.

Several variables in *lp.mk* may be changed before installing LP to customize the system:

Variable	Default Value	Meaning
SPOOL	*/usr/spool/lp*	spool directory
ADMIN	*lp*	logname of LP Administrator

GROUP	*bin*	group owning LP commands/data
ADMDIR	*/usr/lib*	commands of administrator
USRDIR	*/usr/bin*	user commands reside here

If an existing LP spool directory is corrupted (but not the LP programs) or if it needs to be rebuilt from scratch, make sure that **lpsched** is not running and perform the following as superuser:

1. Make copies of any interface programs that are not standard LP software. **DO NOT** make these copies underneath the spool directory. The pathname for printer "p" is */usr/spool/lp/interface/p.*
2. rm −fr /usr/spool/lp
3. Make *−f lp.mk* new. (This recreates the bare LP configuration described above.)

PRECAUTIONS:

1. Some LP commands invoke other LP commands. Moving them after they are built will cause some commands to fail.
2. The files under the SPOOL directory should be modified **only by LP commands.**
3. All LP commands require set-user-id permission. If this is removed, the commands will fail.

CONFIGURING LP—THE "lpadmin" COMMAND

Changes to the LP configuration should be made by using the **lpadmin** command and not by hand. **Lpadmin** will not attempt to alter the LP configuration when **lpsched** is running, except where explicitly noted below.

A. Introducing New Destinations

The following information must be supplied to **lpadmin** when introducing a new printer:

1. The printer name (**−p** printer) is an arbitrary name which must conform to the following rules:
   - It must be no longer than 14 characters.
   - It must consist solely of alphanumeric characters and underscores.
   - It must not be the name of an existing LP destination (printer or class).
2. The device associated with the printer (**−v** device). This is the pathname of a hardwired printer, a login terminal, or other file that is writable by lp.
3. The printer interface program. This may be specified in one of three ways:
   - It may be selected from a list of model interfaces supplied with LP (**−m** model).
   - It may be the same interface that an existing printer uses (**−e** printer).
   - It may be a program supplied by the LP administrator (**−i** interface).

Information which need not always be supplied when creating a new printer includes:

1. The user may specify **−h** to indicate that the device for the printer is hardwired or the device is the name of a file (this is assumed by default). If, on the other hand, the device is the pathname of a login terminal,

then **-l** must be included on the command line. This indicates to *lpsched* that it must automatically disable this printer each time *lpsched* starts running. This fact is reported by *lpstat* when it indicates printer status:

```
$ lpstat -pa
printer a (login terminal) disabled Oct 31 11:15—
        disabled by scheduler: login terminal
```

This is done because device names for login terminals can be (and usually are) associated with different physical devices from day to day. If the scheduler did not take this action, somebody might log in and be surprised that LP is spooling to his/her terminal!

2. The new printer may be added to an existing class or added to a new class (**-c**class). New class names must conform to the same rules for new printer names.

EXAMPLES

The following examples will be referenced by further examples in later sections.

1. Create a printer called pr1 whose device is /dev/printer and whose interface program is the model hp interface:

```
$ /usr/lib/lpadmin -ppr1 -v/dev/printer -mhp
```

2. Add a printer called pr2 whose device is /dev/tty22 and whose interface is a variation of the model prx interface. It is also a login terminal:

```
$ cp /usr/spool/lp/model/prx xxx
  < edit xxx >
$ /usr/lib/lpadmin -ppr2 -v/dev/tty22 -ixxx -l
```

3. Create a printer called pr3 whose device is /dev/tty23. The pr3 will be added to a new class called cl1 and will use the same interface as printer pr2:

```
$ /usr/lib/lpadmin -ppr3 -v/dev/tty23 -epr2 -ccl1
```

B. Modifying Existing Destinations

Modifications to existing destinations must always be made with respect to a printer name (**-p** printer). The modifications may be one or more of the following:

1. The device for the printer may be changed (**-v** device). If this is the only modification, then this may be done even while *lpsched* is running. This facilitates changing devices for login terminals.

2. The printer interface program may be changed (**-m** model, **-e** printer, **-i** interface).

3. The printer may be specified as hardwired (**-h**) or as a login terminal (**-l**).

4. The printer may be added to a new or existing class (**-c**class).

5. The printer may be removed from an existing class (**-r** class). Removing the last remaining member of a class causes the class to be deleted. No destination may be removed if it has pending requests. In that case, **lpmove** or **cancel** should be used to move or delete the pending requests.

EXAMPLES

These examples are based on the LP configuration created by those in the previous section.

1. Add printer pr2 to class cl1:

```
$ /usr/lib/lpadmin -ppr2 -ccl1
```

2. Change pr2's interface program to the model prx interface, change its device to /dev/tty24, and add it to a new class called cl2:

```
$ /usr/lib/lpadmin -ppr2 -mprx -v/dev/tty24 -ccl2
```

   Note that printers pr2 and pr3 now use different interface programs even though pr3 was originally created with the same interface as pr2. Printer pr2 is now a member of two classes.

3. Specify printer pr2 as a hardwired printer:

```
$ /usr/lib/lpadmin -ppr2 -h
```

4. Add printer pr1 to class cl2:

```
$ /usr/lib/lpadmin -ppr1 -ccl2
```

   The members of class cl2 are now pr2 and pr1, in that order. Requests routed to class cl2 will be serviced by pr2 if both pr2 and pr1 are ready to print; otherwise, they will be printed by the one which is next ready to print.

5. Remove printers pr2 and pr3 from class cl1:

```
$ /usr/lib/lpadmin -ppr2 -rcl1
$ /usr/lib/lpadmin -ppr3 -rcl1
```

   Since pr3 was the last remaining member of class cl1, the class is removed.

6. Add pr3 to a new class called cl3.

```
$ /usr/lib/lpadmin -ppr3 -ccl3
```

C. Specifying the System Default Destination

The system default destination may be changed even when **lpsched** is running.

EXAMPLES

1. Establish class cl1 as the system default destination:

```
$ /usr/lib/lpadmin -dcl1
```

2. Establish no default destination:

```
$ /usr/lib/lpadmin -d
```

D. Removing Destinations

Classes and printers may be removed only if there are no pending requests that were routed to them. Pending requests must either be canceled using **cancel** or moved to other destinations using **lpmove** before destinations may be removed. If the removed destination is the system default destination, then the system will have

no default destination until the default destination is respecified. When the last remaining member of a class is removed, then the class is also removed. The removal of a class never implies the removal of printers.

EXAMPLES

1. Make printer pr1 the system default destination:

```
$ /usr/lib/lpadmin -dpr1
```

   Remove printer pr1:

```
$ /usr/lib/lpadmin -xpr1
```

   Now there is no system default destination.

2. Remove printer pr2:

```
$ /usr/lib/lpadmin -xpr2
```

   Class cl2 is also removed since pr2 was its only member.

3. Remove class cl3:

```
$ /usr/lib/lpadmin -xcl3
```

   Class cl3 is removed, but printer pr3 remains.

MAKING AN OUTPUT REQUEST—THE "lp" COMMAND

Once LP destinations have been created, users may request output by using the **lp** command. The request id that is returned may be used to see if the request has been printed or to cancel the request.

The LP program determines the destination of a request by checking the following list in order:

- If the user specifies *-d* dest on the command line, then the request is routed to *dest.*
- If the environment variable **LPDEST** is set, the request is routed to the value of *LPDEST.*
- If there is a system default destination, then the request is routed there.
- Otherwise, the request is rejected.

EXAMPLES

1. There are at least four ways to print the password file on the system default destination:

```
lp /etc/passwd
lp < /etc/passwd
cat /etc/passwd | lp
lp -c /etc/passwd
```

   The last three ways cause copies of the file to be printed, whereas the first way prints the file directly. Thus, if the file is modified between the time the request is made and the time it is actually printed, then the changes will be reflected in the output.

2. Print two copies of file abc on printer xyz and title the output "my file":

```
pr abc | lp -dxyz -n2 -t " my file "
```

3. Print file xxx on a Diablo* 1640 printer called zoo in 12-pitch and write to the user's terminal when printing has completed:

```
lp -dzoo -o12 -w xxx
```

In this example, "12" is an option that is meaningful to the model Diablo 1640 interface program that prints output in 12-pitch mode [see **lpadmin**(1M)].

FINDING LP STATUS—LPSTAT

The **lpstat** command is used to find status information about LP requests, destinations, and the scheduler.

EXAMPLES

1. List the status of all pending output requests made by this user:

```
lpstat
```

The status information for a request includes the request id, the logname of the user, the total number of characters to be printed, and the date and time the request was made.

2. List the status of printers p1 and p2:

```
lpstat -pp1,p2
```

CANCELING REQUESTS—CANCEL

The LP requests may be canceled using the **cancel** command. Two kinds of arguments may be given to the command—request ids and printer names. The requests named by the request ids are canceled and requests that are currently printing on the named printers are canceled. Both types of arguments may be intermixed.

EXAMPLE

Cancel the request that is now printing on printer xyz:

```
cancel xyz
```

If the user that is canceling a request is not the same one that made the request, then mail is sent to the owner of the request. LP allows any user to cancel requests in order to eliminate the need for users to find LP administrators when unusual output should be purged from printers.

ALLOWING AND REFUSING REQUESTS—ACCEPT AND REJECT

When a new destination is created, **lp** will reject requests that are routed to it. When the LP administrator is sure that it is set up correctly, he or she should allow **lp** to accept requests for that destination. The **accept** command performs this function.

Sometimes it is necessary to prevent **lp** from routing requests to destinations. If printers have been removed or are waiting to be repaired or if too many requests are building for printers, then it may be desirable to cause

* Trademark of Diablo Systems, Inc.

lp to reject requests for those destinations. The **reject** command performs this function. After the condition that led to the rejection of requests has been remedied, the **accept** command should be used to allow requests to be taken again.

The acceptance status of destinations is reported by the **-a** option of **lpstat**.

EXAMPLES

1. Cause **lp** to reject requests for destination xyz:

```
/usr/lib/reject -r " printer xyz needs repair "  xyz
```

Any users that try to route requests to xyz will encounter the following:

```
$ lp -dxyz file
lp: can not accept requests for destination " xyz "
        -printer xyz needs repair
```

2. Allow **lp** to accept requests routed to destination xyz:

```
/usr/lib/accept xyz
```

ALLOWING AND INHIBITING PRINTING—ENABLE AND DISABLE

The **enable** command allows the LP scheduler to print requests on printers. That is, the scheduler routes requests only to the interface programs of enabled printers. Note that it is possible to enable a printer but to prevent further requests from being routed to it.

The **disable** command cancels the effects of the **enable** command. It prevents the scheduler from routing requests to printers, independently of whether or not **lp** is allowing them to accept requests. Printers may be disabled for several reasons including malfunctioning hardware, paper jams, and end of day shutdowns. If a printer is busy at the time it is disabled, then the request that it was printing will be reprinted in its entirety either on another printer (if the request was originally routed to a class of printers) or on the same one when the printer is reenabled. The **-c** option causes the currently printing requests on busy printers to be canceled in addition to disabling the printers. This is useful if strange output is causing a printer to behave abnormally.

EXAMPLE

Disable printer xyz because of a paper jam:

```
$ disable -r " paper jam "  xyz
printer " xyz "  now disabled
```

Find the status of printer xyz:

```
$ lpstat -pxyz
printer " xyz "  disabled since Jan 5 10:15 -
        paper jam
```

Now, reenable xyz:

```
$ enable xyz
printer " xyz "  now enabled
```

MOVING REQUESTS BETWEEN DESTINATIONS—LPMOVE

Occasionally, it is useful for LP administrators to move output requests between destinations. For instance, when a printer is down for repairs, it may be desirable to move all of its pending requests to a working printer. This is one way to use the **lpmove** command. The other use of this command is to move specific requests to a different destination. **Lpmove** will refuse to move requests while the LP scheduler is running.

EXAMPLES

1. Move all requests for printer abc to printer xyz:

   ```
   $ /usr/lib/lpmove abc xyz
   ```

   All of the moved requests are renamed from abc-nnn to xyz-nnn. As a side effect, destination abc is no longer accepting further requests.

2. Move requests zoo-543 and abc-1200 to printer xyz:

   ```
   $ /usr/lib/lpmove zoo-543 abc-1200 xyz
   ```

   The two requests are now renamed xyz-543 and xyz-1200.

STOPPING AND STARTING THE SCHEDULER—LPSHUT AND LPSCHED

Lpsched is the program that routes the output requests that were made with **lp** through the appropriate printer interface programs to be printed on line printers. Each time the scheduler routes a request to an interface program, it records an entry in the log file, */usr/spool/lp/log*. This entry contains the logname of the user that made the request, the request id, the name of the printer that the request is being printed on, and the date and time that printing first started. In the case that a request has been restarted, more than one entry in the log file may refer to the request. The scheduler also records error messages in the log file. When **lpsched** is started, it renames */usr/spool/lp/log* to */usr/spool/lp/oldlog* and starts a new log file.

No printing will be performed by the LP system unless **lpsched** is running. Use the command

```
lpstat -r
```

to find the status of the LP scheduler.

Lpsched is normally started by the **/etc/rc** program as described above and continues to run until the UNIX system is shut down. The scheduler operates in the */usr/spool/lp* directory. When it starts running, it will exit immediately if a file called *SCHEDLOCK* exists. Otherwise, it creates this file in order to prevent more than one scheduler from running at the same time.

Occasionally, it is necessary to shut down the scheduler in order to reconfigure LP or to rebuild the LP software. The command

```
/usr/lib/lpshut
```

causes **lpsched** to stop running and terminates all printing activity. All requests that were in the middle of printing will be reprinted in their entirety when the scheduler is restarted.

To restart the LP scheduler, use the command

```
/usr/lib/lpsched
```

Shortly after this command is entered, **lpstat** should report that the scheduler is running. If not, it is possible that a previous invocation of **lpsched** exited without removing *SCHEDLOCK*, so try the following:

```
rm -f /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched
```

The scheduler should be running now.

PRINTER INTERFACE PROGRAMS

Every LP printer must have an interface program which does the actual printing on the device that is currently associated with the printer. Interface programs may be shell procedures, C programs, or any other executable programs. The LP model interfaces are all written as shell procedures and can be found in the */usr/spool/lp/model* directory. At the time **lpsched** routes an output request to a printer P, the interface program for P is invoked in the directory */usr/spool/lp* as follows:

```
interface/P id user title copies options file ...
where
id is the request id returned by lp
user is logname of user who made the request
title is optional title specified by the user
copies is number of copies requested by user
options is a blank-separated list of class or
printer-dependent options specified by user
file is the full pathname of a file to be printed
```

EXAMPLES

The following examples are requests made by user "smith" with a system default destination of printer "xyz". Each example lists an **lp** command line followed by the corresponding command line generated for printer xyz's interface program:

1. lp /etc/passwd /etc/group
   interface/xyz xyz-52 smith " " 1 " " /etc/passwd /etc/group

2. pr /etc/passwd | lp -t " users " -n5
   interface/xyz xyz-53 smith users 5 " "
   /usr/spool/lp/request/xyz/d0-53

3. lp /etc/passwd -oa -ob
   interface/xyz xyz-54 smith " " 1 " a b " /etc/passwd

When the interface program is invoked, its standard input comes from */dev/null* and both the standard output and standard error output are directed to the printer's device. Devices are opened for reading as well as

writing when file modes permit. In the case where a device is a regular file, all output is appended to the end of the file.

Given the command line arguments and the output directed to a device, interface programs may format their output in any way they choose. Interface programs must ensure that the proper stty modes (terminal characteristics such as baud rate, output options, etc.) are in effect on the output device. This may be done as follows in a shell interface only if the device is opened for reading:

```
stty mode ... <&1
```

That is, take the standard input for the **stty** command from the device.

When printing has completed, it is the responsibility of the interface program to exit with a code indicative of the success of the print job. Exit codes are interpreted by **lpsched** as follows:

CODE	MEANING TO LPSCHED
zero	The print job has completed successfully.
1 to 127	A problem was encountered in printing this particular request (e.g., too many nonprintable characters). This problem will not affect future print jobs. **Lpsched** notifies users by mail that there was an error in printing the request.
greater than 127	These codes are reserved for internal use by **lpsched**. Interface programs must not exit with codes in this range.

When problems that are likely to affect future print jobs occur (e.g., a device filter program is missing), the interface programs would be wise to disable printers so that print requests are not lost. When a busy printer is disabled, the interface program will be terminated with signal 15.

SETTING UP HARDWIRED DEVICES AND LOGIN TERMINALS AS LP PRINTERS

A. Hardwired Devices

As an example of how to set up a hardwired device for use as an LP printer, let us consider using tty line 15 as printer xyz. As superuser, perform the following:

1. Avoid unwanted output from non-LP processes and ensure that LP can write to the device:

   ```
   $ chown lp /dev/tty15
   $ chmod 600 /dev/tty15
   ```

2. Change */etc/inittab* so that tty15 is not a login terminal. In other words, ensure that */etc/getty* is not trying to log users in at this terminal. Change the entries for line 15 to:

   ```
   1:15:o:
   2:15:o:
   ```

   Enter the command:

   ```
   $ init 2
   ```

   If there is currently an invocation of */etc/getty* running on tty15, kill it. Now, and when the UNIX system is rebooted, tty15 will be initialized with default stty modes. Thus, it is up to LP interface programs to establish the proper baud rate and other stty modes for correct printing to occur.

3. Introduce printer xyz to LP using the model prx interface program:

   $ /usr/lib/lpadmin -pxyz -v/dev/tty15 -mprx

4. When xyz is created, it will initially be disabled and **lp** will be rejecting requests routed to it. If it is desired, allow **lp** to accept requests for xyz:

   /usr/lib/accept xyz

   This will allow requests to build up for xyz, and to be printed when it is enabled at a later time.

5. When it is desired for printing to occur, be sure that the printer is ready to receive output. For several printers, this means that the top of form has been adjusted and that the printer is on-line. Enable printing to occur on xyz:

   enable xyz

   When requests have been routed to xyz, they will begin printing.

B. Login Terminals

Login terminals may also be used as LP printers. To do this for a Diablo 1640 terminal called abc, perform the following:

1. Introduce printer abc to LP using the model 1640 interface program:

   $ /usr/lib/lpadmin -pabc -v/dev/null -m1640 -l

   Note that */dev/null* is used as abc's device because we will specify the actual device each time that abc is enabled. This device may be different from day to day. When abc is created, it will initially be disabled and **lp** will be rejecting requests routed to it. If it is desired, allow **lp** to accept requests for abc:

   /usr/lib/accept abc

   This will allow requests to build up for abc and to be printed when it is enabled at a later time. It is not advisable to enable abc for printing, however, until the following steps have been taken.

2. Log terminal in if this has not already been done.

3. Assuming the **tty**(1) command reports that this terminal is */dev/tty02*, associate this device with printer abc:

   $ /usr/lib/lpadmin -pabc -v/dev/tty02

   Note that **lpadmin** may be used only by an LPA. If it is desired for other users to routinely perform this step, then an LPA may establish a program owned by **lp** or by **root** with set-user-id permission that performs this function.

4. When it is desired for printing to occur, be sure that the printer is ready to receive output. For several printers, this means that the top of form has been adjusted. Enable printing to occur on abc:

   enable abc

   When requests have been routed to abc, they will begin printing.

5. When all printing has stopped on abc or when you want it back as a regular login terminal, you may prevent it from printing more output:

```
$ disable abc
printer "abc" now disabled
```

If abc is enabled when the UNIX system is rebooted or when **lpsched** is restarted, it will be disabled automatically.

SUMMARY

The administrative functions of the LP administrator have been described in detail. These functions include configuring and reconfiguring LP; maintaining printer interface programs; accepting, rejecting, and moving print requests; stopping and starting the LP scheduler; and enabling and disabling printers. LP offers administrators the following advantages over other centrally supported printer packages:

- Printers may be grouped into classes.
- LP may be configured to meet the needs of each site.
- Administrators may supply interface programs to format output in any way desirable.
- LP functions are performed by simple commands and not by hand.

APPENDIX E: SYSTEM ACTIVITY PACKAGE

This section describes the design and implementation of the UNIX System Activity Package. The UNIX operating system contains a number of counters that are incremented as various system actions occur. The system activity package reports UNIX system-wide measurements including Central Processing Unit(CPU) utilization, disk and tape Input/Output(I/O) activities, terminal device activity, buffer usage, system calls, system switching and swapping, file-access activity, queue activity, and message and semaphore activities. The package provides four commands that generate various types of reports. Procedures that automatically generate daily reports are also included. The five functions of the activity package are:

- **sar**(1) command—allows a user to generate system activity reports in real-time and to save system activities in a file for later usage.
- **sag**(1G) command—displays system activity in a graphical form.
- **sadp**(1) command—samples disk activity once every second during a specified time interval and reports disk usage and seek distance in either tabular or histogram form.
- **timex**(1)—a modified **time**(1) command that times a command and also reports concurrent system activity.
- system activity daily reports—procedures are provided for sampling and saving system activities in a data file periodically and for generating the daily report from the data file.

The following subsections describe the system activity counters located in the operating system kernel; the commands in the system activity package, the procedure for generating daily reports, source file descriptions, and an explanation of some statistics.

SYSTEM ACTIVITY COUNTERS

The system activity counters provide the basis for the system activity reporting system. Most of these counters are described by the **sysinfo** data structure in **/usr/include/sys/sysinfo.h.** The system table overflow counters are in the **_syserr** structure. The device activity counters are extracted from the device status tables. The I/O activity of all disk devices is recorded.

In the following paragraphs, the system activity counters that are sampled by the system activity package are described.

Cpu time counters: There are four time counters that may be incremented at each clock interrupt 60 times per second. Exactly one of the *cpu*[] counters is incremented on each interrupt, according to the mode the CPU is in at the interrupt; idle, user, kernal, and wait for I/O completion.

Lread and lwrite: The *lread* and *lwrite* counters are used to count logical read and write requests issued by the system to block devices.

Bread and bwrite: The *bread* and *bwrite* counters are used to count the number of times data is transferred between the system buffers and the block devices. These actual I/Os are triggered by logical I/Os that

cannot be satisfied by the current contents of the buffers. The ratio of block I/O to logical I/O is a common measure of the effectiveness of the system buffering.

Phread and phwrite: The *phread* and *phwrite* counters count read and write requests issued by the system to raw devices.

Swapin and Swapout: The swapin and swapout counters are incremented by each system request initiating transfer from or to the swap device, including virtual memory page transfers. Frequently used programs are kept on the swap device and swapped in rather than loaded from the file system. The swapin counter reflects these initial loading operations as well as resumptions of activity, while the swapout counter reveals the level of actual "swapping." The amount of data transferred between the swap device and memory is measured in blocks and counted by bswapin and bswapout.

Pswitch and syscall: These counters are related to the management of multiprogramming. *Syscall* is incremented every time a system call is invoked. The numbers of invocations of **read**(2), **write**(2), **fork**(2), and **exec**(2) system calls are kept in counters *sysread, syswrite, sysfork,* and *sysexec,* respectively. *Pswitch* counts the times the switcher was invoked, which occurs when:

a. a system call resulted in a road block

b. an interrupt occurred resulting in awakening a higher priority process

c. 1-second clock interrupt.

Iget, namei, and dirblk: These counters apply to file-access operations. *Iget* and *namei,* in particular, are the names of UNIX operating system routines. The counters record the number of times that the respective routines are called. *Namei* is the routine that performs file system path searches. It searches the various directory files to get the associated i-number of a file corresponding to a special path. *Iget* is a routine called to locate the inode entry of a file (i-number). It first searches the in-core inode table. If the inode entry is not in the table, routine *iget* will get the inode from the file system where the file resides and make an entry in the in-core inode table for the file. *Iget* returns a pointer to this entry. *Namei* calls *iget,* but other file access routines also call *iget.* Therefore, counter *iget* is always greater than counter *namei.*

Counter *dirblk* records the number of directory block reads issued by the system. It is noted that the directory blocks read divided by the number of *namei* calls estimates the average path length of files.

Runque, runocc, swpque, and swpocc: These counters are used to record queue activities. They are implemented in the *clock.c* routine. At every 1 second interval, the clock routine examines the process table to see whether any processes are in core and in ready state. If so, the counter *runocc* is incremented and the number of such processes are added to counter *runque.* While examining the process table, the clock routine also checks whether any processes in the swap device are in ready state. The counter *swpocc* is incremented if the swap queue is occupied, and the number of processes in swap queue is added to counter *swpque.*

Readch and writech: The *readch* and *writech* counters record the total number of bytes (characters) transferred by the **read** and **write** system calls, respectively.

Monitoring terminal device activities: There are six counters monitoring terminal device activities. *Revint, xmtint,* and *mdmint* are counters measuring hardware interrupt occurrences for receiver, transmitter, and modem individually. *Rawch, canch,* and *outch* count number of characters in the raw queue, canonical queue, and output queue. Characters generated by devices operating in the *cooked* mode, such as terminals, are counted in both *rawch* and (as edited) in *canch,* but characters from raw devices, such as communication processors, are counted only in *rawch.*

Msg and sema counters: These counters record message sending and receiving activities and semaphore operations, respectively.

Monitoring I/O activities: As to the I/O activity for a disk or tape device, four counters are kept for each disk or tape drive in the device status table. Counter *io_ops* is incremented when an I/O operation has occurred on the device. It includes block I/O, swap I/O, and physical I/O. *Io_bcnt* counts the amount of data transferred between the device and memory in 512 byte units. *Io_act* and *io_resp* measure the active time and response time of a device in time ticks summed over all I/O requests that have completed for each device. The device active time includes the device seeking, rotating and data transferring times, while the response time of an I/O operation is from the time the I/O request is queued to the device to the time when the I/O completes.

Inodeovf, fileovf, textovf, and procovf: These counters are extracted from *_syserr* structure. When an overflow occurs in any of the inode, file, text and process tables, the corresponding overflow counter is incremented.

System Activity Commands

The system activity package provides three commands for generating various system activity reports and one command for profiling disk activities. These tools facilitate observation of system activity during

- a controlled stand-alone test of a large system
- an uncontrolled run of a program to observe the operating environment
- normal production operation.

Commands **sar** and **sag** permit the user to specify a sampling interval and number of intervals for examining system activity and then to display the observed level of activity in tabular or graphical form. The **timex** command reports the amount of system activity that occurred during the precise period of execution of a timed command. The **sadp** command allows the user to establish a sampling period during which access location and seek distance on specified disks are recorded and later displayed as a tabular summary or as a histogram.

The "sar" command

The **sar** command can be used in the following ways:

- When the frequency arguments **t** and **n** are specified, it invokes the data collection program **sadc** to sample the system activity counters in the operating system every **t** seconds for **n** intervals and generates system activity reports in real-time. Generally, it is desirable to include the option to save the sampled data in a file for later examination. The format of the data file is shown in **sar**(8). In addition to the system counters, a time stamp is also included. It gives the time at which the sample was taken.
- If no frequency arguments are supplied, it generates system activity reports for a specified time interval from an existing data file that was created by **sar** at an earlier time.

A convenient usage is to run **sar** as a background process, saving its samples in a temporary file but sending its standard output to */dev/null*. Then an experiment is conducted after which the system activity is extracted from the temporary file. The **sar**(1) manual entry describes the usage and lists various types of reports. Attachment 11.3 gives formula for deriving each reported item.

The "sag" command

Sag displays system activity data graphically. It relies on the data file produced by a prior run of **sar** after which any column of data or the combination of columns of data of the **sar** report can be plotted. A fairly simple but powerful command syntax allows the specification of cross plots or time plots. Data items are selected using the **sar** column header names. The **sar**(1G) manual entry describes its options and usage. The system activity

graphical program invokes **graphics**(1G) and **tplot**(1G) commands to have the graphical output displayed on any of the terminal types supported by **tplot**.

The "timex" command

The **timex** command is an extension of the **time**(1) command. Without options, **timex** behaves exactly like **time**. In addition to giving the time information, it also prints a system activity report derived from the system counters. The manual entry **timex**(1) explains its usage. It should be emphasized that the user and sys times reported in the second and third lines are for the measured process itself including all its children while the remaining data (including the cpu user % and cpu sys %) are for the entire system.

While the normal use of **timex** will probably be to measure a single command, multiple commands can also be timed; either by combining them in an executable file and timing it, or more concisely, by typing:

```
timex sh -c " cmd1; cmd2; ... ; "
```

This establishes the necessary parent-child relationships to correctly extract the user and system times consumed by **cmd1**,**cmd2**, . . . (and the shell).

The "sadp" command

Sadp is a user level program that can be invoked independently by any user. It requires no storage or extra code in the operating system and allows the user to specify the disks to be monitored. The program is reawakened every second, reads system tables from */dev/kmem*, and extracts the required information. Because of the 1 second sampling, only a small fraction of disk requests are observed; however, comparative studies have shown that the statistical determination of disk locality is adequate when sufficient samples are collected.

In the operating system, there is an *iobuf* for each disk drive. It contains two pointers which are head and tail of the I/O active queue for the device. The actual requests in the queue may be found in three buffer header pools—system buffer headers for block I/O requests, physical buffer headers for physical I/O requests, and swap buffer headers for swap I/O. Each buffer header has a forward pointer which points to the next request in the I/O active queue and a backward pointer which points to the previous request.

Sadp snapshots the *iobuf* of the monitored device and the three buffer header pools once every second during the monitoring period. It then traces the requests in the I/O queue, records the disk access location, and seeks distance in buckets of 8 cylinder increments. At the end of monitoring period, it prints out the sampled data. The output of **sadp** can be used to balance load among disk drives and to rearrange the layout of a particular disk pack. The usage of this command is described in manual entry **sadp**(1).

Daily Report Generation

The previous part described the commands available to users to initiate activity observations. It is probably desirable for each installation to routinely monitor and record system activity in a standard way for historical analysis. This part describes the steps that a system administrator may follow to automatically produce a standard daily report of system activity.

Facilities

- **sadc**—The executable module of **sadc.c** (see Attachment 11.1) which reads system counters from */dev/kmem* and records them to a file. In addition to the file argument, two frequency arguments are usually specified to indicate the sampling interval and number of samples to be taken. In case no frequency arguments are given, it writes a dummy record in the file to indicate a system restart.

- **sa1**—The shell procedure that invokes **sadc** to write system counters in the daily data file */usr/adm/sa* **dd** where **dd** represents the day of the month. It may be invoked with sampling interval and iterations as arguments.

- **sa2**—The shell procedure that invokes the **sar** command to generate daily report */usr/adm/sa/sar**dd*** from the daily data file */usr/adm/sa/sa**dd***. It also removes daily data files and report files after 7 days. The starting and ending times and all report options of **sar** are applicable to **sa2**.

Suggested Operational Setup

It is suggested that the **cron**(1M) control the normal data collection and report generation operations. For example, the sample entries in */usr/lib/crontab:*

```
0 * * * 0,6 su sys -c " /usr/lib/sa/sa1 "
0 18- * * 1-5 su sys -c " /usr/lib/sa/sa1 "
0 8-17 * * 1-5 su sys -c " /usr/lib/sa/sa1 1200 3 "
```

would cause the data collection program **sadc** to be invoked every hour on the hour. Moreover, depending on the arguments presented, it writes data to the data file one to three times at every 20 minutes. Therefore, under the control of **cron**(1M), the data file is written every 20 minutes between 8:00 and 18:00 on weekdays and hourly at other times.

Note that data samples are taken more frequently during prime time on weekdays to make them available for a finer and more detailed graphical display. It is suggested that **sa1** be invoked hourly rather than invoking it once every day; this ensures that if the system crashes data collection will be resumed within an hour after the system is restarted.

Because system activity counters restart from zero when the system is restarted, a special record is written on the data file to reflect this situation. This process is accomplished by invoking **sadc** with no frequency arguments within */etc/rc* when going to multiuser state:

```
su adm -c " /usr/lib/sa/sadc /usr/adm/sa/sa'date +%d' "
```

Cron(1M) also controls the invocation of **sar** to generate the daily report via shell procedure **sa2**. One may choose the time period the daily report is to cover and the groups of system activity to be reported. For instance, if:

```
0 20 * * 1-5 su sys -c " /usr/lib/sa/sa2 -s 8:00 -e 18:00 -i 3600 -uybd "
```

is an entry in */usr/lib/crontab,* **cron** will execute the **sar** command to generate daily reports from the daily data file at 20:00 on weekdays. The daily report reports the CPU utilization, terminal device activity, buffer usage, and device activity every hour from 8:00 to 18:00.

In case of a shortage of the disk space or for any other reason, these data files and report files can be removed by the superuser. The manual entry **sar**(8) describes the daily report generation procedure.

SOURCE FILES

When source code is provided, the following source file and shell programs are in the directory /usr/src/cmd/sa.

sa.h The system activity header file defines the structure of data file and device information for measured devices. It is included in **sadc.c**, **sar.c**, and **timex.c**.

sadc.c The data collection program that accesses */dev/kmem* to read the system activity counters and writes data either on standard output or on a binary data file. It is invoked by the **sar** command generating a real-time report. It is also invoked indirectly by entries in */usr/lib/crontab* to collect system activity data.

sar.c The report generation program invokes **sadc** to examine system activity data, generates reports in real-time, and saves the data to a file for later usage. It may also generate system activity reports from an existing data file. It is invoked indirectly by **cron** to generate daily reports.

saghdr.h The header file for **saga.c** and **sagb.c**. It contains data structures and variables used by **saga.c** and **sagb.c**.

saga.c & sagb.c The graph generation program that first invokes **sar** to format the data of a data file in a tabular form and then displays the **sar** data in graphical form.

sa1.sh The shell procedure that invokes **sadc** to write data file records. It is activated by entries in */usr/lib/crontab*.

sa2.sh The shell procedure that invokes **sar** to generate the report. It also removes the daily data files and daily report files after a week. It is activated by an entry in */usr/lib/crontab* on weekdays.

timex.c The program that times a command and generates a system activity report.

sadp.c The program that samples and reports disk activities.

THE SYSINFO STRUCTURE

Feb 3 15:46 1984 sysinfo.h Page 1

```
/*     Convergent Technologies - System V - May 1983    */
/*     "@(#)sysinfo.h  1.2"     */

#ifndef sysinfo_h
#define sysinfo_h

#include <sys/types.h>

struct sysinfo {
        time_t  cpu[4];
#define         CPU_IDLE        0
#define         CPU_USER        1
#define         CPU_KERNAL      2
#define         CPU_WAIT        3
        time_t  wait[3];
#define         W_IO    0
#define         W_SWAP  1
#define         W_PIO   2
        long    bread;
        long    bwrite;
        long    lread;
        long    lwrite;
        long    phread;
        long    phwrite;
        long    swapin;
        long    swapout;
        long    bswapin;
        long    bswapout;
        long    pswitch;
        long    syscall;
        long    sysread;
        long    syswrite;
        long    sysfork;
        long    sysexec;
        long    runque;
        long    runocc;
        long    swpque;
        long    swpocc;
        long    iget;
        long    namei;
        long    dirblk;
        long    readch;
        long    writech;
        long    rcvint;
        long    xmtint;
        long    mdmint;
        long    rawch;
        long    canch;
        long    outch;
        long    msg;
        long    sema;
};
```

```
extern struct sysinfo sysinfo;

struct syswait {
```



```
        short   iowait;
        short   swap;
        short   physio;
};

extern struct syswait syswait;

struct syserr {
        long    inodeovf;
        long    fileovf;
        long    textovf;
        long    procovf;
        long    sbi[5];
#define         SBI_SILOC       0
#define         SBI_CRDRDS      1
#define         SBI_ALERT       2
#define         SBI_FAULT       3
#define         SBI_TIMEO       4
};

extern struct syserr syserr;
#endif
```

DERIVATION OF BASIC STATISTICS

Here is how the basic system activity statistics are derived. Each item discussed below is the data difference sampled at two distinct times, t2 and t1.

CPU Utilization

%-of-cpu-x = cpu-x / (cpu-idle + cpu-user + cpu-kernel + cpu-wait) * 100

where cpu-x is cpu-idle, cpu-user, cpu-kernel (cpu-sys), or cpu-wait.

Cached Hit Ratio

%-of-cached-I/O = (logical-I/O − block-I/O) / logical-I/O * 100

where cached I/O is cached read or cached write.

Disk or Tape I/O Activity

%-of-busy = I/O-active / (t2 − t1) * 100;
avg-queue-length = I/O-resp / I/O-active;
avg-wait = (I/O-resp − I/O-active) / I/O-ops;
avg-service-time = I/O-active / I/O-ops.

Queue Activity

avg-x-queue-length = x-queue / x-queue-occupied-time;
%-of-x-queue-occupied-time = x-queue-occupied-time / (t2 − t1);

where x-queue is run queue or swap queue.

The Rest of System Activity

avg-rate-of-x = x / (t2 − t1)

where x is swap in/out, blks swapped in/out, terminal device activities, read/write characters, block read/write, logical read/write, process switch, system calls, read/write, fork/exec, iget, namei, directory blocks read, disk/tape I/O activities, message or semaphore activities.

Appendix F
TM3Ø Keyboard Translation Table

Table F-1 shows the ASCII character set for the TM3Ø. The characters in the left column are hexadecimal digits from Ø (zero) to F. The middle column shows the keystrokes that generate the ASCII code. The column on the right gives the characters, if any, that appear on screen when you press the key(s) shown in the middle column.

Table F-1. TM3Ø Keyboard Translation Table

Hexadecimal Code	TM3Ø Keyboard Keystrokes	Displayable Characters
ØØ	Ctrl =	No
Ø1	Ctrl A	No
Ø2	Ctrl B	No
Ø3	Ctrl C	No
Ø4	Ctrl D	No
Ø5	Ctrl E	No
Ø6	Ctrl F	No
Ø7	Ctrl G	No
Ø8	<-- (left arrow, bcksp)	No
	or Ctrl H	No
Ø9	Tab or Ctrl I	No
ØA	Ctrl Accept or Ctrl J	No
ØB	Ctrl K	No
ØC	Ctrl L	No
ØD	Return or Ctrl M	No
ØE	Ctrl N	No
ØF	Ctrl O	No
1Ø	Ctrl P	No
11	Ctrl Q	No
12	Ctrl R	No
13	Ctrl S	No
14	Ctrl T	No
15	Ctrl U	No
16	Ctrl V	No
17	Ctrl W	No
18	Ctrl X	No
19	Ctrl Y	No
1A	Ctrl Z	No
1B	(See "Hex Code/Escape Sequences," below)	
1C	Ctrl , (comma)	No
1D	Ctrl {	No
1E	Ctrl . (period)	No
1F	Ctrl [	No
2Ø	Space bar	No
21	!	!

Table F-1. TM3Ø Keyboard Translation Table (Cont.)

Hexadecimal Code	TM3Ø Keyboard Keystrokes	Displayable Characters
52	Shift R	R
53	Shift S	S
54	Shift T	T
55	Shift U	U
56	Shift V	V
57	Shift W	W
58	Shift X	X
59	Shift Y	Y
5A	Shift Z	Z
5B	[	[
5C	Ctrl /	\
5D	]	]
5E		
5F	_ (underscore)	
6Ø	Ctrl '	`
61	A	a
62	B	b
63	C	c
64	D	d
65	E	e
66	F	f
67	G	g
68	H	h
69	I	i
6A	J	j
6B	K	k
6C	L	l
6D	M	m
6E	N	n
6F	O	o
7Ø	P	p
71	Q	q
72	R	r
73	S	s
74	T	t
75	U	u
76	V	v
77	W	w
78	X	x
79	Y	y
7A	Z	z
7B	{	{
7C	|	|
7D	}	}
7E	^	^
7F	Ctrl -	No

Table F-1. TM30 Keyboard Translation Table (Cont.)

Hex Code/ Escape Sequence	TM30 Keyboard Keystrokes	Displayable Characters
ESC [%	- (Numeric Island)	No
ESC [|	F5	No
ESC [&	Ins	No
ESC [-	—> (Right Arrow)	No
ESC [{	F4	No
ESC [}	F6	No
ESC [[	F9	No
ESC [ ]	Delete	No
ESC ['	. (Numeric Island)	No
ESC ["	4 (Numeric Island)	No
ESC [\	Cmd	No
ESC [2 b	Shift 1/2	No
ESC [2 g	Shift Exit	No
ESC [2 h	Shift Erase	No
ESC [2 i	Shift 1 (Numeric Island)	No
ESC [2 j	Shift 2 (Numeric Island)	No
ESC [2 k	Shift 3 (Numeric Island)	No
ESC [2 l	Shift Accept	No
ESC [2 n	Shift Copy	No
ESC [2 o	Shift 0 (Numeric Island)	No
ESC [2 r	Shift Char Attr	No
ESC [2 s	Shift Undo	No
ESC [2 t	Shift Home	No
ESC [2 u	Shift Up Arrow	No
ESC [2 v	Shift Help	No
ESC [2 w	Shift Windo	No
ESC [2 x	Shift Print	No
ESC [2 y	Shift F1	No
ESC [2 z	Shift F3	No
ESC [2 I	Shift F2	No
ESC [2 O	Shift F8	No
ESC [2 Q	Shift Reset	No
ESC [2 R	Shift <— (Lft Arrw, Bsp)	No
ESC [2 U	Shift Slct	No
ESC [2 V	Shift Move	No
ESC [2 W	Shift 7 (Numeric Island)	No
ESC [2 X	Shift 8 (Numeric Island)	No
ESC [2 Y	Shift 9 (Numeric Island)	No
ESC [2 Z	Shift Alt	No
ESC [2	Shift Down Arrow	No
ESC [2 ~	Shift F7	No
ESC [2 #	Shift 5 (Numeric Island)	No
ESC [2 $	Shift 6 (Numeric Island)	No
ESC [2 %	Shift - (Numeric Island)	No
ESC [2 |	Shift F5	No
ESC [2 &	Shift Ins	No
ESC [2 -	Shift —>	No
ESC [2 {	Shift F4	No

USER'S COMMENTS

System 6300 Administrator's Guide
S6000-50-1A

HELP!

Help us help you! Please take the time to complete this form and send it to us. If you do, you may see some of your own contributions in the next manual you obtain from us.

- Does this manual provide the information you need? ☐Yes ☐No
  – What is missing?

- Is the manual accurate? ☐Yes ☐No
  – What is incorrect? (Be specific.)

- Is the manual written clearly? ☐Yes ☐No
  – What is unclear?

- What other comments can you make about this manual?

- What do you like about this manual?

- On a scale of 1 to 10, how do you rate this manual? Low ⊢+-+-+-+-+-+-+-+-⊣ High
  1 2 3 4 5 6 7 8 9 10
- Was this manual difficult to obtain? ☐Yes ☐No

Please include your name and address if you would like a reply.

Name _____
Company _____
Address _____
_____

No postage required if mailed within the USA.

- What is your occupation?

☐ Programmer ☐ Operator ☐ Manager
☐ Systems Analyst ☐ Instructor ☐ Customer Engineer
☐ Engineer ☐ Student ☐ Other _____

- How do you use this manual?

☐ Reference Manual ☐ Introduction to the Subject
☐ In a Class ☐ Introduction to the System
☐ Self Study ☐ Other _____

fold fold

FIRST CLASS
Permit No. 194
Cupertino,
California

BUSINESS REPLY MAIL
No Postage Necessary if Mailed in the United States

Postage will be Paid by . . .

MOTOROLA INC.
10700 N. De Anza Blvd.
Cupertino, CA 95104

Attention: Technical Services, MS 42-1C8

fold fold

Staple Here

B1847A

C ing Here

System 6300 Administrator's Guide

MOTOROLA

Information Systems

S6000-50-1A

10700 North De Anza Blvd., Cupertino, California 95014 • (408) 255-0900